

CREATE THE TAG HANDLER

Similar to JavaBeans, custom tags provide a way for you to easily work with complex Java code in your JSP pages. You can create your own custom tags to suit your needs.

Using custom tags can help make the code in a JSP page easier to work with by allowing you to separate the Java code from the HTML code. Since custom tags can be used in multiple JSP pages, using custom tags also saves you from having to retype the same Java code over and over.

The first step in creating a custom tag is to create a tag handler class file, which stores the methods that perform specific actions when the custom tags are processed. A tag handler must import the `javax.servlet.jsp` and `javax.servlet.jsp.tagext` packages in order to access the classes found in these packages. You can refer to the Java SDK documentation for more information about these packages. A tag handler may need to extend the classes found in these packages, such as the `TagSupport`

class of the `javax.servlet.jsp.tagext` package. For information about extending a class, see page 54.

A simple tag handler must have a `doStartTag` method that contains the code to be executed when the start tag of the custom tag is processed. The access modifier of the `doStartTag` method must be specified as `public` in order for the method to be accessed as part of a tag handler.

The `doStartTag` method must return a value to indicate whether the custom tag will include information that must be processed between the start and end tags, called a body. A body is not required for a simple tag handler, so the `doStartTag` method returns the value `SKIP_BODY`. `SKIP_BODY` is a constant and is defined in the imported `javax` packages. `SKIP_BODY` contains an integer value so the return type of the `doStartTag` method must be specified as `int`.

Extra

Before creating tag handlers, you need to install the Java Servlet API class files, which contain the packages that must be imported when you compile the Java code used to create tag handlers. The Java Servlet API class files are available at the java.sun.com/products/servlet Web site. Make sure you store the Java Servlet API class files in the appropriate directory on your computer. For example, on the Windows platform, the Java Servlet API class files are stored in the `c:\jdk1.3\jre\lib\ext` directory. You should check the Java Servlet API specification documentation for installation instructions specific to your operating system.

Custom tags can enable specialization when developing a Web site. For example, Web page designers can work with the HTML content of a JSP page, while programmers develop the Java code that will make the Web page dynamic. This allows both types of professionals to concentrate on their own areas of expertise.

In addition to the value `SKIP_BODY`, the `doStartTag` method may return two other values. The value `EVAL_BODY_INCLUDE` is returned when the body of the custom tag needs to be processed. If the body of the custom tag must be processed using a `BodyContent` object, the value `EVAL_BODY_TAG` is returned.

CREATE THE TAG HANDLER

```

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

{
}

```

```

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class SimpleTag
{
}

```

```

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class SimpleTag extends javax.servlet.jsp.tagext.TagSupport
{
}

```

```

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class SimpleTag extends javax.servlet.jsp.tagext.TagSupport
{
    public int doStartTag()
    {
        return SKIP_BODY;
    }
}

```

- 1 To import the `javax.servlet.jsp` package, type **import javax.servlet.jsp.***.
- 2 To import the `javax.servlet.jsp.tagext` package, type **import javax.servlet.jsp.tagext.***.
- 3 To create a class for the custom tag, type **public class** followed by a name for the class.
- 4 To extend the `TagSupport` class, type **extends javax.servlet.jsp.tagext.TagSupport**.
- 5 To create the `doStartTag` method, type **public int doStartTag()**.
- 6 In the body of the `doStartTag` method, type **return SKIP_BODY**.

CONTINUED

CREATE THE TAG HANDLER

A simple tag handler can be created to generate a message, such as copyright information or a greeting, which you want to display on several JSP pages in your Web site. You can then simply insert a custom tag into each JSP page where you want to display the message. Using a custom tag can make it easier to update the message in each JSP page where it is used.

After you create the tag handler class and declare the `doStartTag` method, you can use methods of the `PageContext` object to generate output for the tag. The `PageContext` object is used by an object to determine the kind of environment in which the object is contained. Once the object's environment is determined, the tag handler will use the `PageContext` object to help perform the requested actions.

The `getOut` method of the `PageContext` object determines the method being used to send information to a client. For a tag handler that displays a message, you can use the `print` method of the `PageContext` object to generate a text message that the JSP page will send to a Web browser when the custom tag is processed. To display the information for the tag, you use the `print` method and the `getOut` method of the `PageContext` object together.

Using the `print` method of the `PageContext` object may throw an `IOException` error. To handle any errors that occur, you should enclose the code that generates a message in a `try` block and create a `catch` block to catch any exception errors. For more information about error handling, see pages 174 to 185.

Extra

The tag handler class file you create must be stored in a specific directory on your Web server. For example, if you are using the `c:\tomcat\webapps\examples` directory to store your JSP pages on a Windows platform using the Tomcat Web server, you would store your tag handler class files in the `c:\tomcat\webapps\examples\WEB-INF\classes` directory. You can refer to your Web server documentation to determine the proper directory to store your tag handler class files.

The `doStartTag` method is called when the start tag of a custom tag is processed. The start tag is the opening tag. For example, in the HTML code `<title>My Web Page</title>`, the start tag is `<title>` and the end tag is `</title>`. The phrase "My Web Page" is the body of the tag.

When a tag does not require a body, the start and end tags can be combined into one tag in a JSP page.

Example:

```
<mytag></mytag>
```

Can be written as:

```
<mytag />
```

CREATE THE TAG HANDLER (CONTINUED)

```
Untitled - Notepad
File Edit Search Help
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class SimpleTag extends javax.servlet.jsp.tagext.TagSupport
{
    public int doStartTag() throws JspException
    {
        return SKIP_BODY;
    }
}
```

7 To specify that the `doStartTag` method may throw a JSP related exception error, type **throws JspException** on the same line as the method declaration.

```
Untitled - Notepad
File Edit Search Help
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class SimpleTag extends javax.servlet.jsp.tagext.TagSupport
{
    public int doStartTag() throws JspException
    {
        try
        {
            pageContext.getOut().print();
        }

        return SKIP_BODY;
    }
}
```

8 Type the code that creates a `try` block in the body of the `doStartTag` method.

9 In the body of the `try` block, type **pageContext.getOut().print()**.

```
Untitled - Notepad
File Edit Search Help
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class SimpleTag extends javax.servlet.jsp.tagext.TagSupport
{
    public int doStartTag() throws JspException
    {
        try
        {
            pageContext.getOut().print("Welcome to my Web site.");
        }

        return SKIP_BODY;
    }
}
```

10 Between the second set of parentheses, type the message you want the tag to display, enclosed in quotation marks.

```
Untitled - Notepad
File Edit Search Help
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class SimpleTag extends javax.servlet.jsp.tagext.TagSupport
{
    public int doStartTag() throws JspException
    {
        try
        {
            pageContext.getOut().print("Welcome to my Web site.");
        }
        catch (Exception e)
        {
            throw new JspTagException(e.getMessage());
        }
        return SKIP_BODY;
    }
}
```

11 On the line immediately following the `try` block, type the code that creates a `catch` block.

12 In the body of the `catch` block, type **throw new JspTagException(e.getMessage())**.

13 Save the file with the `.java` extension and then compile the source code for the file.

14 Copy the compiled class file to the appropriate directory on your Web server.

You can now add the tag handler class to a tag library descriptor file.

CREATE THE TAG LIBRARY DESCRIPTOR FILE

Once you have created a tag handler for a custom tag, you must create a Tag Library Descriptor (TLD) file that will tell the Web server where to locate the tag handler when the custom tag is used in a JSP page. Tag library descriptor files are XML documents that can be created and edited using a simple text editor.

When creating a tag library descriptor file, you must begin the file with an XML header that specifies information about the file. The XML header consists of the `<?xml?>` tag and the `<!DOCTYPE>` tag. The information in the XML header is a standard requirement of XML documents and will be the same for every tag library descriptor file you create. For more information about XML documents, visit the www.xml.org Web site.

The `<taglib>` and `</taglib>` tags are used to enclose the main body of a tag library descriptor file. There are several tags you must use in the main body of the file. For example,

you must use the `<jspversion>` tag to specify the version of JavaServer Pages that the tag library descriptor file uses.

A tag library descriptor file can contain information about multiple custom tags. To provide information about a custom tag you are creating, use the `<tag>` and `</tag>` tags to enclose the information. To specify the name of a custom tag, use the `<name>` tag. The name you specify must be the same as the name you will use for the tag in a JSP page. To specify the name of the tag handler class file for a custom tag, use the `<tagclass>` tag. You do not need to include the `.class` extension when specifying the name.

Tag library descriptor files should be saved with the `.tld` extension. When using version 3.1 of the Tomcat Web server, the name and location of the tag library descriptor file must be specified in the `web.xml` configuration file. To configure the `web.xml` configuration file, see page 204.

Extra

The following tags must also be included in the main body of all the tag library descriptor files you create.

TAG:	DESCRIPTION:
<code><tlibversion></code>	The version of the tag library descriptor file.
<code><shortname></code>	A name that will be used to reference the tag library descriptor file from a JSP page.
<code><info></code>	A description of the tag library descriptor file.

If you are not using version 3.1 of the Tomcat Web server, you should use the `<uri>` tag in the main body of the tag library descriptor file to provide a unique identifier for the file, such as `<uri>www.maran.com/taglib</uri>` or `<uri>MyTagLibrary</uri>`. A JSP page that uses the tag library will contain the same identifier, linking the JSP page to the tag library descriptor file. In this situation, you do not need to configure the `web.xml` file.

The tag handler class file for a custom tag may be stored in a package. Grouping tags into packages is a common practice that allows you to easily organize and work with a large number of tags. When using the `<tagclass>` tag to specify the name of a tag handler class file that is stored in a package, prefix the name of the class file with the name of the package, separating the names with a dot. For example, a tag handler class file called `SimpleTag.class` that is part of the `mytags.web.text` package can be indicated in the tag descriptor file using the code `<tagclass>mytags.web.text.SimpleTag</tagclass>`.

CREATE THE TAG LIBRARY DESCRIPTOR FILE

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

1 To create the XML header for the tag library descriptor file, type `<?xml version="1.0" encoding="ISO-8859-1" ?>`.

2 Type `<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN" "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">`.

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

3 To create the main body of the tag library descriptor file, type `<taglib>`.

4 Type `<tlibversion>` followed by the version number of the tag library descriptor file. Then type `</tlibversion>`.

5 Type `<jspversion>` followed by the version number of JavaServer Pages required to use the tag library descriptor file. Then type `</jspversion>`.

```
Untitled - Notepad
File Edit Search Help
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
"http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>MyFirstTag</shortname>
  <info>My first tag library descriptor file</info>
  <tag>
    <name>SimpleTag</name>
```

6 Type `<shortname>` followed by a name that will reference the tag library descriptor file from a JSP page. Then type `</shortname>`.

7 Type `<info>` followed by a description of the tag library descriptor file. Then type `</info>`.

8 Type `<tag>` to begin specifying information about a custom tag you are creating.

9 Type `<name>` followed by the name of the custom tag. Then type `</name>`.

```
Untitled - Notepad
File Edit Search Help
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
"http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>MyFirstTag</shortname>
  <info>My first tag library descriptor file</info>
  <tag>
    <name>SimpleTag</name>
    <tagclass>SimpleTag</tagclass>
  </tag>
</taglib>
```

10 Type `<tagclass>` followed by the name of the tag handler class file. Then type `</tagclass>`.

11 Type `</tag>` to complete the information about the custom tag.

12 Type `</taglib>` to complete the main body of the tag library descriptor file.

13 Save the file with the `.tld` extension.

You can now configure the `web.xml` file to work with your tag library descriptor file.

CONFIGURE THE WEB.XML FILE

If you are using version 3.1 of the Tomcat Web server, you must configure a Web application's web.xml file before using custom tags in the application. This task can be completed once you have created the tag handler class and the tag library descriptor file.

Each Web application on your Web server may have its own web.xml file. The web.xml files may be automatically created by the Web server. If you are storing your JSP pages in the \webapps\examples directory under the main Tomcat directory, then the web.xml file you should configure is located in the \webapps\examples\WEB-INF directory under the main Tomcat directory.

The web.xml file defines the setup of certain features of your Web application, such as session tracking and the file name of the default home page. To enable JSP pages in your Web application to use custom tags, you must use

the <taglib> tag to configure the web.xml file. If the web.xml file already contains <taglib> tags, you can add a new <taglib> tag immediately following the existing <taglib> tags. If there are currently no <taglib> tags in the web.xml file, the new <taglib> tag can be inserted anywhere between the <web-app> and </web-app> tags. You can consult the Web server documentation for more information about the web.xml file.

Within the <taglib> and </taglib> tags, you must include the <taglib-uri> and <taglib-location> tags. The <taglib-uri> tag specifies a label, which will be used in your JSP pages to refer to the tag library descriptor file. The label can be an address, such as www.xyzcorp.com, or a word, such as mytags. The <taglib-location> tag specifies the location of the tag library descriptor file on the Web server.

Extra

You must configure the appropriate web.xml file in order to use custom tags with version 3.1 of the Tomcat Web server. Other Web servers, as well as other versions of the Tomcat Web server, may not require any alteration of the configuration file. Carefully consult your Web server documentation to determine what, if any, changes need to be made to the server's configuration files prior to using custom tags.

The web.xml file conforms to the XML specification for document structure and, therefore, should be very easy to read and edit. Text information, such as the label and location of the tag descriptor file, does not have to be enclosed in quotation marks.

In addition to the specific web.xml files for each application on your Web server, there is also a default web.xml file located in the \conf directory within the main Tomcat directory. This default web.xml file stores information such as the server's basic configuration. If you want all your JSP pages in any Web application to have access to a custom tag, you can add the <taglib> tag for the custom tag to the default web.xml file. Both the default web.xml file and the application-specific web.xml file will be accessed by a JSP page that uses a custom tag.

CONFIGURE THE WEB.XML FILE

```

<taglib>
  <taglib-uri>
    http://java.apache.org/tomcat/examples-taglib
  </taglib-uri>
  <taglib-location>
    /WEB-INF/jsp/example-taglib.tld
  </taglib-location>
</taglib>
<taglib>
</taglib>

<security-constraint>
  <web-resource-collection>
    <web-resource-name>Protected Area</web-resource-name>
    <!-- Define the context-relative URL(s) to be protected -->
    <url-pattern>/jsp/security/protected/*</url-pattern>
    <!-- If you list http methods, only those methods are protected -->
    <http-method>DELETE</http-method>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
    <http-method>PUT</http-method>
  </web-resource-collection>
  <auth-constraint>
    <!-- Anyone with one of the listed roles may access this area -->
    <role-name>tomcat</role-name>
    <role-name>role1</role-name>
  </auth-constraint>
</security-constraint>

```

1 Open the web.xml file you want to configure in a text editor.

2 To enable the Web application to use custom tags, type <taglib>.

3 If the web.xml file already contains <taglib> tags, place the new <taglib> tag on the line immediately following the existing <taglib> tags.

```

<taglib>
  <taglib-uri>
    http://java.apache.org/tomcat/examples-taglib
  </taglib-uri>
  <taglib-location>
    /WEB-INF/jsp/example-taglib.tld
  </taglib-location>
</taglib>
<taglib>
  <taglib-uri>
    http://www.maran.com/taglib
  </taglib-uri>
</taglib>

<security-constraint>
  <web-resource-collection>
    <web-resource-name>Protected Area</web-resource-name>
    <!-- Define the context-relative URL(s) to be protected -->
    <url-pattern>/jsp/security/protected/*</url-pattern>
    <!-- If you list http methods, only those methods are protected -->
    <http-method>DELETE</http-method>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
    <http-method>PUT</http-method>
  </web-resource-collection>
  <auth-constraint>
    <!-- Anyone with one of the listed roles may access this area -->
    <role-name>tomcat</role-name>
    <role-name>role1</role-name>
  </auth-constraint>
</security-constraint>

```

3 To create a label for the tag library descriptor file, type <taglib-uri>.

4 Type the label for the tag library descriptor file.

5 Type </taglib-uri> to close the <taglib-uri> tag.

```

<taglib>
  <taglib-uri>
    http://java.apache.org/tomcat/examples-taglib
  </taglib-uri>
  <taglib-location>
    /WEB-INF/jsp/example-taglib.tld
  </taglib-location>
</taglib>
<taglib>
  <taglib-uri>
    http://www.maran.com/taglib
  </taglib-uri>
  <taglib-location>
    /WEB-INF/taglib.tld
  </taglib-location>
</taglib>

<security-constraint>
  <web-resource-collection>
    <web-resource-name>Protected Area</web-resource-name>
    <!-- Define the context-relative URL(s) to be protected -->
    <url-pattern>/jsp/security/protected/*</url-pattern>
    <!-- If you list http methods, only those methods are protected -->
    <http-method>DELETE</http-method>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <!-- Anyone with one of the listed roles may access this area -->
    <role-name>tomcat</role-name>
    <role-name>role1</role-name>
  </auth-constraint>
</security-constraint>

```

6 To specify the location of the tag library descriptor file, type <taglib-location>.

7 Type the location of the tag library descriptor file.

8 Type </taglib-location> to close the <taglib-location> tag.

```

<taglib>
  <taglib-uri>
    http://java.apache.org/tomcat/examples-taglib
  </taglib-uri>
  <taglib-location>
    /WEB-INF/jsp/example-taglib.tld
  </taglib-location>
</taglib>
<taglib>
  <taglib-uri>
    http://www.maran.com/taglib
  </taglib-uri>
  <taglib-location>
    /WEB-INF/taglib.tld
  </taglib-location>
</taglib>

<security-constraint>
  <web-resource-collection>
    <web-resource-name>Protected Area</web-resource-name>
    <!-- Define the context-relative URL(s) to be protected -->
    <url-pattern>/jsp/security/protected/*</url-pattern>
    <!-- If you list http methods, only those methods are protected -->
    <http-method>DELETE</http-method>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <!-- Anyone with one of the listed roles may access this area -->
    <role-name>tomcat</role-name>
    <role-name>role1</role-name>
  </auth-constraint>
</security-constraint>

```

9 Type </taglib> to close the <taglib> tag.

10 Save the web.xml file.

You can now use the custom tag in your JSP pages.

USING A CUSTOM TAG

Once you have compiled the tag handler class file, created the tag library descriptor file and configured the web.xml file for a custom tag, you can use the custom tag in a JSP page.

You must first include the `taglib` directive in the JSP page to identify the tag library descriptor file that contains the tag you want to use. To add the `taglib` directive to a JSP page, you place the directive statement between the `<%@` opening delimiter and the `%>` closing delimiter.

The `taglib` directive uses the `uri` attribute to specify an identifier for the tag library descriptor file. The identifier must be the same identifier specified for the tag library descriptor file in the web.xml file.

Within the `taglib` directive, you must also specify a prefix you want to use to reference the tag library that contains the custom tag information. Each tag library requires a different

prefix, so you can use different prefixes to work with custom tags that have the same name but are stored in different tag libraries.

To use a custom tag in a JSP page, you type the prefix and the name you assigned to the tag in the tag library descriptor file, separated by a colon. Like HTML tags, the start tag is enclosed in angle brackets. The end tag is also enclosed in angle brackets and begins with a forward slash. For simple tags that do not contain a body, or information between the start and end tags, the start and end tags can be combined into one tag, such as `<myTags:SimpleTag />`.

Extra

The custom tag examples in this chapter were created using Tomcat Web server version 3.1. This version of Tomcat uses a label, known as the URI, to map to the tag library descriptor file. Other Web servers, as well as other versions of the Tomcat Web server, may require a different format when using the `taglib` directive. You should consult your Web server documentation for information about using the `taglib` directive.

If any information in the required supporting files for the custom tag, such as the web.xml file, is incorrect, an error may be generated when the `taglib` directive or the custom tag is processed by the Web server. To prevent problems with the JSP page, you should ensure that the tag handler class file includes error handling processes, such as a `try` block and a `catch` block.

Custom tags that display information can be enclosed within HTML tags or other custom tags that will format or organize the information the tag will display.

Example:

```
<b><myTags:SimpleTag /></b>
```

Result:

Welcome to my Web site.

USING A CUSTOM TAG

```
<html>
<head>
<title>Tag Example</title>
</head>
<body>

<%@ taglib uri="http://www.maran.com/taglib" %>

</body>
</html>
```

1 To specify the location of the tag library descriptor file for the custom tag you want to use in the JSP page, type `<%@ taglib uri="" %>`.

2 Between the quotation marks, type the identifier of the tag library descriptor file.

```
<html>
<head>
<title>Tag Example</title>
</head>
<body>

<%@ taglib uri="http://www.maran.com/taglib" prefix="myTags" %>

</body>
</html>
```

3 To specify the prefix you want to use for the tag library, type `prefix=""`.

4 Between the quotation marks, type the prefix you want to use.

```
<html>
<head>
<title>Tag Example</title>
</head>
<body>

<%@ taglib uri="http://www.maran.com/taglib" prefix="myTags" %>
<myTags:SimpleTag</myTags:SimpleTag>
</body>
</html>
```

5 To use the custom tag, type `<>`.

6 Between the angle brackets, type the prefix you specified in the `taglib` directive, followed by a colon. Then type the name you specified for the custom tag in the tag library descriptor file.

7 To close the custom tag, type `</>`. Then repeat step 6.

```
Tag Example - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Home Search Favorites History
Address http://127.0.0.1:8080/examples/usetag.jsp Go Links
Welcome to my Web site.
```

8 Save the page with the `.jsp` extension and then display the JSP page in a Web browser.

The Web browser displays the result of using a custom tag.

CREATE THE TAG HANDLER FOR A TAG WITH AN ATTRIBUTE

A custom tag can be set up to support attributes that are specified when the tag is used in a JSP page. This adds flexibility to the tag and allows you to customize the tag's behavior. For example, you can have a tag that displays a heading accept an attribute that specifies the color of the heading.

An attribute for a custom tag is represented by a variable in a tag handler class file. When you create the variable that will represent an attribute, you can assign a default value to the variable. The tag handler will use this value for the attribute if a value is not specified when the custom tag is used.

When the custom tag is used with an attribute in the JSP page, the value specified for the attribute is passed to the tag handler as a variable. In order to convert a value specified for an attribute in a JSP page to a variable, you must use a setter method. The access modifier of a setter method must be set

to `public` and the return type set to `void`, since the method does not return a value. The name of the method is the same as the name of the variable that stores the value for the attribute, but begins with a capital letter and is prefixed by the word `set`. The parentheses at the end of the setter method name enclose the data type of the variable and the variable used to pass the value to the method. The argument is then assigned to the variable used to store the attribute value.

The variable that represents the attribute value is typically declared in the class body of the tag handler. This allows the variable to be accessed by any method in the tag handler class file.

Apply It

You can create multiple attributes for each tag. You must create a variable and a setter method for each attribute.

Example:

```
private String message = "No message specified.";
private String boldText = "b";
public int doStartTag() throws JspException
{
    try
    {
        String output="<" + boldText + ">" + message + "</" +
        boldText + ">";
        pageContext.getOut().print(output);
    }
    catch(Exception e)
    {
        throw new JspTagException(e.getMessage());
    }
    return SKIP_BODY;
}
public void setMessage(String text)
{
    message = text;
}
public void setBoldText(String text)
{
    boldText = text;
}
```

CREATE THE TAG HANDLER FOR A TAG WITH AN ATTRIBUTE

```
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class AttributeTag extends javax.servlet.jsp.tagext.TagSupport
{
    public int doStartTag() throws JspException
    {
        try
        {
            pageContext.getOut().print(message);
        }
        catch (Exception e)
        {
            throw new JspTagException(e.getMessage());
        }
        return SKIP_BODY;
    }
}
```

1 Type the code that imports the `javax.servlet.jsp` and the `javax.servlet.jsp.tagext` packages.

2 Type the code that creates a class for a custom tag.

3 Type the code that creates a `doStartTag` method.

```
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class AttributeTag extends javax.servlet.jsp.tagext.TagSupport
{
    private String message = "No message specified.";

    public int doStartTag() throws JspException
    {
        try
        {
            pageContext.getOut().print(message);
        }
        catch (Exception e)
        {
            throw new JspTagException(e.getMessage());
        }
        return SKIP_BODY;
    }
}
```

4 To create a variable to represent an attribute that will be specified in a JSP page, type `private String` followed by a name for the variable. Then type `= ""`.

5 Between the quotation marks, type a default value for the variable.

```
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class AttributeTag extends javax.servlet.jsp.tagext.TagSupport
{
    private String message = "No message specified.";

    public int doStartTag() throws JspException
    {
        try
        {
            pageContext.getOut().print(message);
        }
        catch (Exception e)
        {
            throw new JspTagException(e.getMessage());
        }
        return SKIP_BODY;
    }

    public void setMessage(String text)
    {
    }
}
```

6 To declare a setter method that will convert the value of the attribute to the variable you created in step 4, type `public void`.

7 To name the method, type `set` immediately followed by the name of the variable, beginning with a capital letter. Then type `()`.

8 Between the parentheses, type the data type of the variable followed by a name for the variable that will pass the value to the setter method.

```
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class AttributeTag extends javax.servlet.jsp.tagext.TagSupport
{
    private String message = "No message specified.";

    public int doStartTag() throws JspException
    {
        try
        {
            pageContext.getOut().print(message);
        }
        catch (Exception e)
        {
            throw new JspTagException(e.getMessage());
        }
        return SKIP_BODY;
    }

    public void setMessage(String text)
    {
        message = text;
    }
}
```

9 Type the name of the variable that represents the attribute followed by `=` and the variable you specified in step 8. Enclose the code in braces.

10 Save the file with the `.java` extension and then compile the source code for the file.

11 Copy the compiled class file to the appropriate directory on your Web server.

You can now add the tag handler class to a tag library descriptor file.

CREATE THE TAG LIBRARY DESCRIPTOR FILE FOR A TAG WITH AN ATTRIBUTE

You need to add an `<attribute>` tag to a tag library descriptor file for each attribute of a custom tag. An `<attribute>` tag allows you to specify details about an attribute of a custom tag and is placed following the `<tagclass>` tag in a tag library descriptor file.

The name of the attribute is specified using the `<name>` tag, which is placed following the `<attribute>` tag. The name specified with the `<name>` tag must match the name that will be used when the custom tag is used on a JSP page. The name of the attribute specified with the `<name>` tag is case sensitive. Most attribute names use only lower case letters.

After the name of the attribute is specified, a `<required>` tag is used to specify if the attribute is required when the custom tag is used on a JSP page. If a value of `false` is specified for a `<required>` tag, the use of the attribute

is optional when the custom tag is used. If a value of `true` is specified, the attribute must be included each time the custom tag is used.

When you specify that an attribute is not required, you should ensure that the tag handler class file for the custom tag contains the code that specifies a default value for the attribute in the event that the attribute is left out when the custom tag is used. This code can be part of the method in the tag handler that is used to set the value of the attribute when the attribute is included when the custom tag is used. For more information about creating a tag handler for a tag with an attribute, see page 208.

Extra

You can specify as many attributes as required by your custom tag. To add an additional attribute, you can simply use another set of `<attribute>` tags.

Example:

```
<tag>
  <name>AttributeTag</name>
  <tagclass>AttributeTag</tagclass>
  <attribute>
    <name>message</name>
    <required>false</required>
  </attribute>
  <attribute>
    <name>length</name>
    <required>true</required>
  </attribute>
</tag>
```

When using a custom tag with an attribute, you can have the value of the attribute be determined at runtime by a section of JSP code, such as an expression, by adding an `<rtexprvalue>` tag and assigning it a value of `true`.

Example:

```
<tag>
  <name>AttributeTag</name>
  <tagclass>AttributeTag</tagclass>
  <attribute>
    <name>message</name>
    <required>false</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
</tag>
```

CREATE THE TAG LIBRARY DESCRIPTOR FILE FOR A TAG WITH AN ATTRIBUTE

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

<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>MyFirstTag</shortname>
  <info>My first tag library descriptor file</info>
  <tag>
    <name>AttributeTag</name>
    <tagclass>AttributeTag</tagclass>
    <attribute>
```

1 Type code that creates a tag library descriptor file. See page 202 for information about creating a tag library descriptor file.

2 To specify an attribute for a custom tag, type `<attribute>` following the `<tagclass>` tag.

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

<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>MyFirstTag</shortname>
  <info>My first tag library descriptor file</info>
  <tag>
    <name>AttributeTag</name>
    <tagclass>AttributeTag</tagclass>
    <attribute>
      <name>message</name>
```

3 To specify the name of the attribute, type `<name>` followed by the name of the attribute. Then type `</name>`.

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

<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>MyFirstTag</shortname>
  <info>My first tag library descriptor file</info>
  <tag>
    <name>AttributeTag</name>
    <tagclass>AttributeTag</tagclass>
    <attribute>
      <name>message</name>
      <required>false</required>
```

4 To specify if the attribute is required when the custom tag is used, type `<required>` followed by `true` or `false`. Then type `</required>`.

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

<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>MyFirstTag</shortname>
  <info>My first tag library descriptor file</info>
  <tag>
    <name>AttributeTag</name>
    <tagclass>AttributeTag</tagclass>
    <attribute>
      <name>message</name>
      <required>false</required>
    </attribute>
  </tag>
</taglib>
```

5 Type `</attribute>` to end the attribute specification.

6 Save the tag library descriptor file with a `.tld` extension.

You can now use the custom tag with an attribute.

USING A CUSTOM TAG WITH AN ATTRIBUTE

To use a custom tag with an attribute, you must include a `taglib` directive in the JSP page to identify the tag library descriptor file that contains the tag you want to use.

As with any other custom tag that you wish to use in a JSP page, you must specify the prefix and the name you assigned to the tag in the tag library descriptor file, separated by a colon. To add an attribute to the custom tag, you type the name of the attribute and a value to be assigned to the attribute, separated by an equal sign. The value of the attribute must be enclosed in quotation marks.

When using custom tags with an attribute, you can use the common notation, which uses both a start and an end tag, or the shortened notation, which combines the start and end tags into one tag. When using the common notation,

the attribute must be specified in the start tag of the custom tag. When using the shortened notation, the attribute should be specified within the tag, such as `<mt:AttributeTag message="Welcome to my Web page." />`.

The tag library descriptor file indicates whether an attribute is required or not. For information about specifying required attributes in the tag library descriptor file, see page 210. If an attribute is optional, you should ensure that the tag handler class file can process the custom tag when an attribute is not specified. This may be done by assigning a default value to be used in case the attribute is not specified in the custom tag. For information about assigning default attribute values in the tag handler class file, see page 208.

Extra

Attribute values can be enclosed within single or double quotation marks.

Example:

```
<mt:AttributeTag message="This is a tag with an attribute" />
```

Can be typed as:

```
<mt:AttributeTag message='This is a tag with an attribute' />
```

If the `<rtexprvalue>` tag is set to `true` in the tag library descriptor file, you can use a JSP expression in the JSP page to determine the value of an attribute. This allows you to include dynamically generated attribute values in your custom tags. If the `<rtexprvalue>` tag is not specified or is set to `false`, then the attribute value must be a string.

Example:

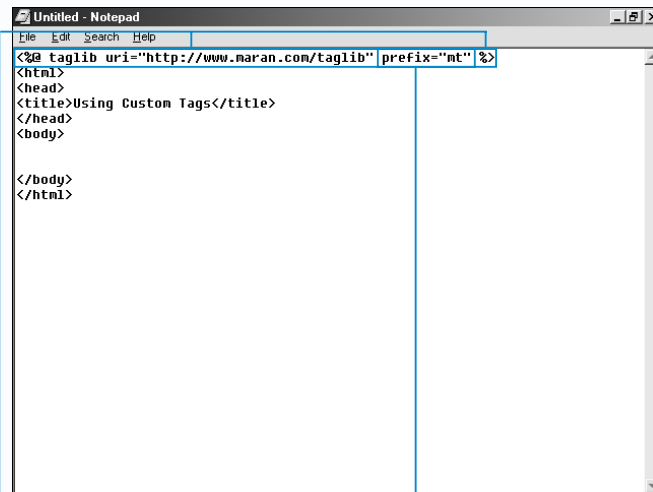
```
<mt:AttributeTag message="<%= Session.getAttribute("userName") %>" />
```

You can use more than one attribute in a tag if the tag handler class file and the tag library descriptor file support multiple attributes.

Example:

```
<mt:AttributeTag message="Welcome to my Web page" encloseText="h1" />
```

USING A CUSTOM TAG WITH AN ATTRIBUTE



```

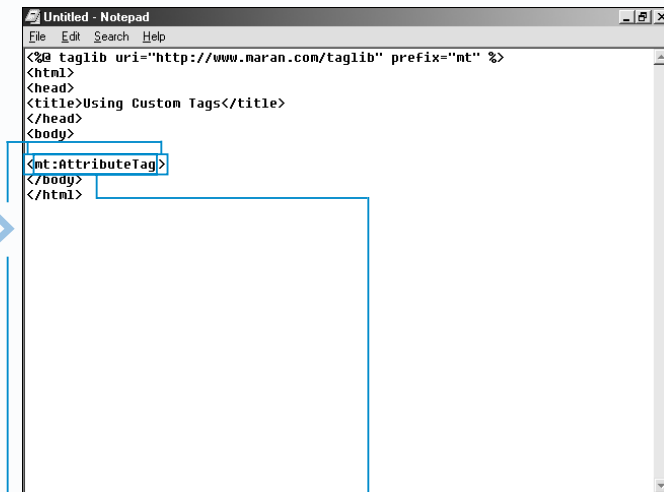
<%@ taglib uri="http://www.maran.com/taglib" prefix="mt" %>
<html>
<head>
<title>Using Custom Tags</title>
</head>
<body>

</body>
</html>

```

1 Create the `taglib` directive that specifies the location of the tag library descriptor file for the custom tag you want to use in the JSP page.

2 In the `taglib` directive, type the code that specifies the prefix you want to use for the tag library.



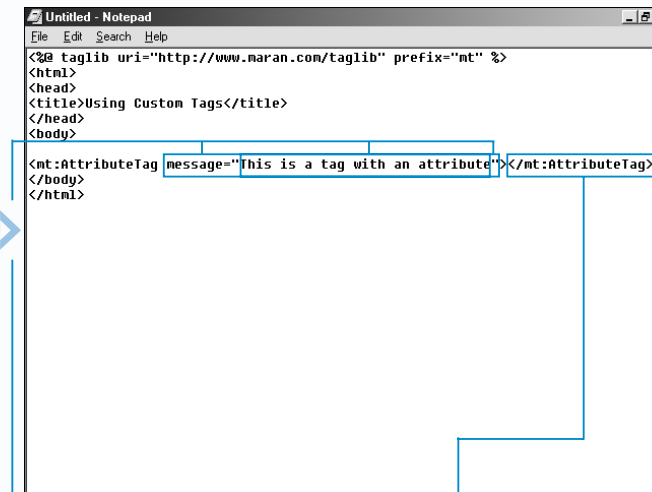
```

<%@ taglib uri="http://www.maran.com/taglib" prefix="mt" %>
<html>
<head>
<title>Using Custom Tags</title>
</head>
<body>
<mt:AttributeTag>
</body>
</html>

```

3 To use the custom tag, type `<>`.

4 Between the angle brackets, type the prefix you specified in the `taglib` directive, followed by a colon. Then type the name of the custom tag.



```

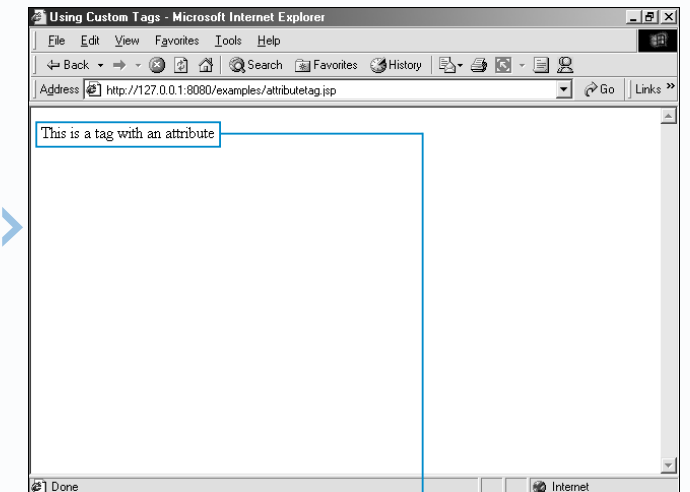
<%@ taglib uri="http://www.maran.com/taglib" prefix="mt" %>
<html>
<head>
<title>Using Custom Tags</title>
</head>
<body>
<mt:AttributeTag message="This is a tag with an attribute"></mt:AttributeTag>
</body>
</html>

```

5 To add an attribute to the custom tag, type the name of the attribute followed by `=`.

6 Between the quotation marks, type the value to be assigned to the attribute.

7 To close the custom tag, type `</>`. Then repeat step 4.



This is a tag with an attribute

8 Save the page with the `.jsp` extension and then display the JSP page in a Web browser.

The Web browser displays the result of using a custom tag with an attribute.

CREATE THE TAG HANDLER FOR A TAG WITH A BODY

The body of a tag is the information enclosed within the start and end tags. The tag body can consist of plain text or any JSP code, including scriptlets, expressions and directives. You can create a tag handler class file to use the information contained in the body of a custom tag. The tag handler class file required for a custom tag with a body is similar to that of a simple custom tag.

As with a tag handler class for a simple tag, a `doStartTag` method must be included. The `doStartTag` method is processed when the start tag of a custom tag is encountered in a JSP page. The `doStartTag` method of a tag handler for a tag with a body must return the value `EVAL_BODY_INCLUDE`, which instructs the Web server to process the information contained in the tag body. When the `doStartTag` method has finished processing and the

`EVAL_BODY_INCLUDE` value has been returned, the Web server includes the information in the tag body in the results that are sent to the client.

The tag handler class file for a custom tag with a body should also include a method called `doEndTag`. The `doEndTag` method contains the code to be executed after the body is processed. This method is executed when the end tag of the custom tag is encountered in the JSP page. The `doEndTag` method must also return a value. The return value determines whether or not the remainder of the JSP page must be processed. In most cases, you will use the `EVAL_PAGE` return value, which indicates to the Web server that the rest of the JSP page should be processed. If you want the Web server to stop processing the JSP page and ignore the remainder of the code in the JSP page, use the `SKIP_PAGE` return value.

Apply It

You can create versatile custom tags that can use attributes and the information in the tag body at the same time. For example, you can modify a custom tag to apply a header level specified by an attribute of the tag to the information in the body of the tag.

Example:

```
private String level = "2";
public int doStartTag() throws JspException
{
    try
    {
        pageContext.getOut().print("<h" + level + "> * * ");
    }
    catch (Exception e)
    {
        throw new JspTagException(e.getMessage());
    }
    return EVAL_BODY_INCLUDE;
}
public int doEndTag() throws JspException
{
    try
    {
        pageContext.getOut().print(" * * </h" + level + ">");
    }
    catch (Exception e)
    {
        throw new JspTagException(e.getMessage());
    }
    return EVAL_PAGE;
}
public void setLevel(String text)
{
    level = text;
}
```

CREATE THE TAG HANDLER FOR A TAG WITH A BODY

```
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class SimpleBodyTag extends javax.servlet.jsp.tagext.TagSupport
{
    public int doStartTag() throws JspException
    {
        try
        {
            pageContext.getOut().print("<h2> * * ");
        }
        catch (Exception e)
        {
            throw new JspTagException(e.getMessage());
        }
        return EVAL_BODY_INCLUDE;
    }
}
```

1 Type the code that imports the `javax.servlet.jsp` and the `javax.servlet.jsp.tagext` packages.

2 Type the code that creates a class for a custom tag.

3 Type the code that creates a `doStartTag` method.

4 In the body of the `doStartTag` method, type `return EVAL_BODY_INCLUDE` to allow the tag to process the information in the tag body.

```
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class SimpleBodyTag extends javax.servlet.jsp.tagext.TagSupport
{
    public int doStartTag() throws JspException
    {
        try
        {
            pageContext.getOut().print("<h2> * * ");
        }
        catch (Exception e)
        {
            throw new JspTagException(e.getMessage());
        }
        return EVAL_BODY_INCLUDE;
    }

    public int doEndTag() throws JspException
    {
        try
        {
            pageContext.getOut().print(" * * </h2>");
        }
    }
}
```

5 To create the `doEndTag` method and specify that it may throw a JSP related exception error, type `public int doEndTag() throws JspException`.

6 Type the code that creates a `try` block in the body of the `doEndTag` method.

7 In the body of the `try` block, type the code that performs an action after the tag body is processed.

```
public int doStartTag() throws JspException
{
    try
    {
        pageContext.getOut().print("<h2> * * ");
    }
    catch (Exception e)
    {
        throw new JspTagException(e.getMessage());
    }
    return EVAL_BODY_INCLUDE;
}

public int doEndTag() throws JspException
{
    try
    {
        pageContext.getOut().print(" * * </h2>");
    }
    catch (Exception e)
    {
        throw new JspTagException(e.getMessage());
    }
}
```

8 On the line immediately following the `try` block, type the code that creates a `catch` block that throws a `JspTagException` exception.

```
public int doStartTag() throws JspException
{
    try
    {
        pageContext.getOut().print("<h2> * * ");
    }
    catch (Exception e)
    {
        throw new JspTagException(e.getMessage());
    }
    return EVAL_BODY_INCLUDE;
}

public int doEndTag() throws JspException
{
    try
    {
        pageContext.getOut().print(" * * </h2>");
    }
    catch (Exception e)
    {
        throw new JspTagException(e.getMessage());
    }
    return EVAL_PAGE;
}
```

9 In the body of the `doEndTag` method, type `return` followed by the return value that specifies if the Web server should process the remainder of the JSP page.

10 Save the file with the `.java` extension and then compile the source code for the file.

11 Copy the compiled class file to the appropriate directory on your Web server.

You can now add the tag handler class to a tag library descriptor file.

CREATE THE TAG LIBRARY DESCRIPTOR FILE FOR A TAG WITH A BODY

After you create a tag handler class file for a custom tag, you should include an entry in the tag library descriptor file that indicates whether the tag will include a body. The body of a custom tag is the information that is enclosed by the start and end tags of the custom tag.

You use the `<bodycontent>` tag in the tag library descriptor file to indicate that a custom tag will contain a body. Using the `<bodycontent>` tag does not usually affect how a custom tag operates, since the processing of the tag is performed mainly by the tag handler. The `<bodycontent>` tag is used primarily for providing information about the custom tag itself. You should always include a `<bodycontent>` tag, especially if you are creating tag libraries that you intend to share with other people.

In the tag library descriptor file, the `<bodycontent>` tag must be located between the `<tag>` and `</tag>` tags that contain detailed information about the custom tag. Only one body content entry can exist for each custom tag.

You specify the content type of the custom tag by inserting a value between the `<bodycontent>` and `</bodycontent>` tags. If no content type is specified, the custom tag will assume the default value of `JSP`, which allows the custom tag to use JSP code as the body of the tag. It is good programming practice to specify `JSP` as the content type for any custom tag that will use a body.

If you are using version 3.1 of the Tomcat Web server, you must specify the name and location of the tag library descriptor file for a custom tag with a body in the `web.xml` configuration file. For more information, see page 204.

Extra

To be consistent in your code, you should include a `<bodycontent>` tag in the tag library descriptor file even for custom tags you create or update that do not include a body. You can specify that a tag does not use a body by indicating the value `empty` as the content type of the tag.

Example:

```
<bodycontent>empty</bodycontent>
```

When you specify `JSP` as the content type of the `<bodycontent>` tag, you are not restricted to using only JSP code in the body of the tag. You can also use HTML tags, plain text, other custom tags and any other valid Web page content in the body of your custom tag.

If you want to use non-JSP information, such as an SQL statement, as the body of your custom tag, you should specify the value `tagdependent` as the content type of the tag. When using the `tagdependent` content type, you must ensure that the code in the tag handler class file is capable of properly interpreting the body content specified in the custom tag.

Example:

```
<bodycontent>tagdependent</bodycontent>
```

CREATE THE TAG LIBRARY DESCRIPTOR FILE FOR A TAG WITH A BODY

```

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

<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>MyFirstTag</shortname>
  <info>My first tag library descriptor file</info>
  <tag>
    <name>SimpleBodyTag</name>
    <tagclass>SimpleBodyTag</tagclass>
    <bodycontent>
  </tag>
</taglib>

```

1 Type the code that creates a tag library descriptor file. See page 202 for information about creating a tag library descriptor file.

2 To specify that the custom tag will include a body, type `<bodycontent>` following the `<tagclass>` tag.

```

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

<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>MyFirstTag</shortname>
  <info>My first tag library descriptor file</info>
  <tag>
    <name>SimpleBodyTag</name>
    <tagclass>SimpleBodyTag</tagclass>
    <bodycontent>JSP
  </tag>
</taglib>

```

3 To specify the content type of the tag body, enter the content type the tag will use.

```

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

<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>MyFirstTag</shortname>
  <info>My first tag library descriptor file</info>
  <tag>
    <name>SimpleBodyTag</name>
    <tagclass>SimpleBodyTag</tagclass>
    <bodycontent>JSP</bodycontent>
  </tag>
</taglib>

```

4 Type `</bodycontent>` to end the body content specification.

```

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

<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>MyFirstTag</shortname>
  <info>My first tag library descriptor file</info>
  <tag>
    <name>SimpleBodyTag</name>
    <tagclass>SimpleBodyTag</tagclass>
    <bodycontent>JSP</bodycontent>
  </tag>
</taglib>

```

5 Save the tag library descriptor file with the `.tld` extension.

You can now use the custom tag with a body.

USING A CUSTOM TAG WITH A BODY

After a tag handler that uses information contained in the body of a tag has been created, the tag library descriptor file has been configured to indicate that the custom tag will process the body and the web.xml file has been configured for the Tomcat Web server if necessary, the custom tag that makes use of a body can be employed on a JSP page.

As with all custom tags, the `taglib` directive must be placed in the JSP page before the custom tag can be used. The `uri` attribute and a prefix you want to use to reference the custom tag that uses a body must be specified in the `taglib` directive.

Using a custom tag with a body makes it easy to generate data that will surround the information found in the body. For example, a custom tag can create specific HTML tags

to enclose the body of the tag specified in the JSP page. You can use a custom tag to apply simple formatting options, such as bolding or changing the font of text, to text supplied in the body of the tag. A custom tag can also be used to place the information in the body within more complex HTML structures, such as tables or lists.

You should not use the short form of writing a tag for a tag that processes a body, since the tag must use start and end tags to denote the start and end of the body content. If the short form of writing a tag is used for a custom tag that uses a body, an empty body will be passed to the tag handler.

Extra

The information that is placed in the body of a custom tag does not have to be on one line. If the tag handler is simply returning the body of the custom tag to the JSP page, the information in the source code of the JSP page will have the same format as the information in the body of the tag.

Example:

```
<pre>
<myTag:SimpleBodyTag>
Welcome
  To
    My
      Web Page
</myTag:SimpleBodyTag>
</pre>
```

Since the body is returned directly to the JSP page by the tag handler, the body of a custom tag can also include valid HTML code.

TYPE THIS:

```
<myTag:SimpleBodyTag><i>Welcome To My Web Page</i></myTag:SimpleBodyTag>
```

RESULT:

```
** Welcome to My Web Page **
```

JSP expressions and scriptlets can also be placed in the body of a custom tag. The JSP code will be evaluated and the information generated by the JSP code will be passed to the tag handler.

TYPE THIS:

```
<myTag:SimpleBodyTag><%= new java.util.Date() %></myTag:SimpleBodyTag>
```

RESULT:

```
** Wed Apr 18 12:00:00 EST 2001 **
```

USING A CUSTOM TAG WITH A BODY

- 1 Create the `taglib` directive that specifies the location of the tag library descriptor file for the custom tag you want to use in the JSP page.
- 2 In the `taglib` directive, type the code that specifies the prefix you want to use for the tag library.
- 3 To use the custom tag, type `<>`.
- 4 Between the angle brackets, type the prefix you specified in the `taglib` directive, followed by a colon. Then type the name of the custom tag.
- 5 Type the information you want to use as the body of the custom tag.
- 6 To close the custom tag, type `</>`. Then repeat step 4.
- 7 Save the page with the `.jsp` extension and then display the JSP page in a Web browser.

The Web browser displays the result of using a custom tag with a body.

CREATE THE TAG HANDLER FOR A TAG THAT MANIPULATES A BODY

A tag handler that receives a body passed from a custom tag may simply return the body back to a JSP page without making any changes. You can, however, create a tag handler for a custom tag that can manipulate a body and then return the manipulated body back to a JSP page. For example, a tag handler may change the formatting of the text within the body or use the information in a body to perform other tasks, such as retrieving information from a database.

A tag handler used to manipulate the body of a custom tag must extend the `BodyTagSupport` class, which in turn extends the `TagSupport` class and contains special methods used for processing the body of a custom tag.

A `doAfterBody` method must be used to process the body of a custom tag. Within the `doAfterBody` method, the `getBodyContent` method is used to create a

`BodyContent` object. This object stores information about the body of the custom tag passed to the tag handler.

The `getString` method of the `BodyContent` object is used to retrieve the body and returns a string that can be assigned to a variable. The string variable can then be manipulated. For example, you can use the `toUpperCase` method of the `String` object to convert all the text in the string to uppercase. When generating output to be passed back to a JSP page, the `getEnclosingWriter` method of the `BodyContent` object is used.

You should include a `try` block and `catch` block in the `doAfterBody` method to catch any exception errors that may occur. After the `doAfterBody` method has finished manipulating the body of a custom tag, the method should return the value `SKIP_BODY`.

CREATE THE TAG HANDLER FOR A TAG THAT MANIPULATES A BODY

```

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class MakeUpperCase extends BodyTagSupport
{
    public int doAfterBody()
    {
        BodyContent bodyFromTag = getBodyContent();
    }
}

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class MakeUpperCase extends BodyTagSupport
{
    public int doAfterBody()
    {
        try
        {
            BodyContent bodyFromTag = getBodyContent();
            String bodyText = bodyFromTag.getString();
            bodyFromTag.getEnclosingWriter().print(bodyText.toUpperCase());
        }
        catch (Exception e)
        {
            System.out.println("An error occurred in the tag handler");
        }
    }
}

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class MakeUpperCase extends BodyTagSupport
{
    public int doAfterBody()
    {
        try
        {
            BodyContent bodyFromTag = getBodyContent();
            String bodyText = bodyFromTag.getString();
            bodyFromTag.getEnclosingWriter().print(bodyText.toUpperCase());
        }
        catch (Exception e)
        {
            System.out.println("An error occurred in the tag handler");
        }
        return SKIP_BODY;
    }
}

```

- Type the code that imports the `javax.servlet.jsp` and the `javax.servlet.jsp.tagext` packages.
- Type the code that creates a class for a custom tag, followed by `extends BodyTagSupport`.
- To create the `doAfterBody` method, type `public int doAfterBody()`.
- To create a `BodyContent` object that will store information about the body, type `BodyContent` followed by a name for the `BodyContent` object.
- Type `= getBodyContent()`.
- Type the code that creates a `try` block in the body of the `doAfterBody` method.
- Type the code that creates a string variable that will store the content of the body.
- Type `=` followed by the name of the `BodyContent` object. Then type `.getString()`.
- To display the result of manipulating the body, type the name of the `BodyContent` object followed by `.getEnclosingWriter().print()`.
- Between the second set of parentheses, type the code that manipulates the body.
- On the line immediately following the `try` block, type the code that creates a `catch` block.
- In the body of the `catch` block, type `System.out.println("")`.
- Between the quotation marks, type the information you want to display if an error occurs.
- On the line immediately following the `catch` block, type `return SKIP_BODY`.
- Save the file with the `.java` extension and then compile the source code for the file.
- Copy the compiled class file to the appropriate directory on your Web server.

You can now add the tag handler class to a tag library descriptor file. For more information see page 216.

Extra

In addition to the `doAfterBody` method, other methods, such as the `doStartTag` and `doEndtag` methods, can be used to perform certain actions, such as producing HTML code that is processed before and after the body is manipulated. Since the body of the custom tag is processed using a `BodyContent` object, the return value of the `doStartTag` method should be `EVAL_BODY_TAG`.

Example:

```

public int doStartTag() throws JspException
{
    try
    {
        pageContext.getOut().print("The following text " +
            " will be in uppercase.<br>");
    }
    catch (Exception e)
    {
        throw new JspTagException(e.getMessage());
    }
    return EVAL_BODY_TAG;
}

public int doEndTag() throws JspException
{
    try
    {
        pageContext.getOut().print("<hr>Thank You");
    }
    catch (Exception e)
    {
        throw new JspTagException(e.getMessage());
    }
    return EVAL_PAGE;
}

```

USING A CUSTOM TAG THAT MANIPULATES A BODY

Using a custom tag that manipulates the content of a body is similar to using any other custom tag that contains information between the start and end tags. The difference is that the tag handler can modify the body before it is returned to the JSP page. The information returned to a JSP page depends on the process contained in the tag handler class itself. A tag handler may simply change the formatting of the text within the body or may take the information in a body and use it to perform other tasks, such as retrieving information from a database or sending an e-mail message.

It may take some time for a tag handler that manipulates the body of a custom tag to process the information, such as when using a tag handler to retrieve information from a database. If you are using a custom tag that may take some time to process, it is important to thoroughly test the tag

to evaluate how the tag will perform when used on a Web site under real-world conditions. While a custom tag may perform well under developmental conditions, it may easily malfunction when multiple users of a Web site use the custom tag, as is common when different users are trying to access the same database. If an error occurs within a tag handler while it processes the body content, a server error may be generated and the remainder of the JSP page may not be processed.

The body of a custom tag can be either plain text or be generated by other methods, such as Java code contained within a scriptlet or an expression.

As with other custom tags and tag handlers, changes to the tag library descriptor file and the web.xml file may need to be made before the custom tag can be used.

Apply It

Although a custom tag is usually inserted into the HTML portion of a JSP page, it is possible to re-use a custom tag using JSP code by integrating scriptlets with the custom tag.

TYPE THIS:

```
<%
String[] days = {"mon", "tue", "wed", "thr", "fri", "sat", "sun"};
for (int x = 0; x <= 6; x++)
{
%>
<myTag:MakeUpperCase><%= days[x] %></myTag:MakeUpperCase><br>
<%
}
%>
```

RESULT:

```
MON
TUE
WED
THR
FRI
SAT
SUN
```

USING A CUSTOM TAG THAT MANIPULATES A BODY

```
Untitled - Notepad
File Edit Search Help
<%@ taglib uri="http://www.maran.com/taglib" prefix="myTag" %>
<html>
<head>
<title>Uppercase Text</title>
</head>
<body>
<h2>
</h2>
</body>
</html>
```

1 Create the taglib directive that specifies the location of the tag library descriptor file for the custom tag you want to use in the JSP page.

2 In the taglib directive, type the code that specifies the prefix you want to use for the tag library.

```
Untitled - Notepad
File Edit Search Help
<%@ taglib uri="http://www.maran.com/taglib" prefix="myTag" %>
<html>
<head>
<title>Uppercase Text</title>
</head>
<body>
<h2>
<myTag:MakeUpperCase>
</h2>
</body>
</html>
```

3 To use the custom tag, type <>.

4 Between the angle brackets, type the prefix you specified in the taglib directive, followed by a colon. Then type the name of the custom tag.

```
Untitled - Notepad
File Edit Search Help
<%@ taglib uri="http://www.maran.com/taglib" prefix="myTag" %>
<html>
<head>
<title>Uppercase Text</title>
</head>
<body>
<h2>
<myTag:MakeUpperCase>welcome to my web site</myTag:MakeUpperCase>
</h2>
</body>
</html>
```

5 Type the information you want to use as the body of the custom tag.

6 To close the custom tag, type </>. Then repeat step 4.

```
Uppercase Text - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Home Search Favorites History
Address http://127.0.0.1:8080/examples/manipulatebody.jsp
WELCOME TO MY WEB SITE
```

7 Save the page with the .jsp extension and then display the JSP page in a Web browser.

8 The Web browser displays the result of using a custom tag that manipulates a body.

USING A CUSTOM TAG TO ACCESS A DATABASE

While custom tags are often used for relatively simple tasks, such as formatting text, they are also ideally suited for performing more complex tasks, such as retrieving information from a database.

You must first create a tag handler for the custom tag. The tag handler for a tag that accesses a database is very similar to the tag handler for a custom tag that manipulates a body. For more information, see page 220. To use the body of a custom tag to locate information in a database, you include the same Java code used to work with databases from within a JSP page in the tag handler class file. For information about working with databases, see pages 142 to 153.

You must load the driver for the database, create a connection to the database and create a result set that stores the information you will access in the database. When you use the custom tag in a JSP page, the tag handler retrieves

the information from the database and then uses the `getEnclosingWriter` method of the `BodyContent` object to send the retrieved information back to the JSP page.

When using a tag handler to work with a database, you can place most of your code in the body of the `doAfterBody` method that processes the body of the custom tag. If you are using other methods in the custom tag, you can place parts of the code in the body of the other methods. For example, you can place the code that opens a connection to the database in the body of the `doStartTag` method.

To prevent problems when the JSP page is processed, you need to include `try` and `catch` blocks to handle any errors that may occur.

Apply It

You can use a custom tag that accesses a database as you would use a custom tag that manipulates a body. In your JSP page, you must include the `taglib` directive to specify the `uri` attribute and the `prefix` you want to use to reference the custom tag. In the body of the tag, you should include the information that specifies the data you want to retrieve from the database.

TYPE THIS:

```
<@ taglib uri="http://www.maran.com/taglib" prefix="myTag" %>
<html>
<head>
<title>Phone Numbers</title>
</head>
<body bgcolor="#FFFFFF">
<b>Phone Numbers</b><br>
<myTag:Phone>Hannah</myTag:Phone>
<br>
<myTag:Phone>Paul</myTag:Phone>
</body>
</html>
```

RESULT:

Phone Numbers
Hannah is at extension 678
Paul is at extension 456

USING A CUSTOM TAG TO ACCESS A DATABASE

```
import java.sql.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class Phone extends BodyTagSupport
{
    public int doAfterBody()
    {
        String phoneNum = "";
        return SKIP_BODY;
    }
}
```

1 Type the code that imports the packages required by the custom tag.

2 Type the code that creates the tag handler class and the `doAfterBody` method.

3 In the body of the method, type `return SKIP_BODY`.

4 Type the code that creates a variable that will store the value you want to retrieve from the database.

```
import java.sql.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class Phone extends BodyTagSupport
{
    public int doAfterBody()
    {
        String phoneNum = "";
        BodyContent bodyFromTag = getBodyContent();
        String bodyText = bodyFromTag.getString();

        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        return SKIP_BODY;
    }
}
```

5 To create a `BodyContent` object, type `BodyContent` followed by a name for the object. Then type `= getBodyContent()`.

6 Type the code that creates a string variable that will store the content of the body.

7 To retrieve the content of the body, type `=` followed by the name of the `BodyContent` object. Then type `.getString()`.

8 Type the code that loads the bridge driver.

```
import java.sql.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class Phone extends BodyTagSupport
{
    public int doAfterBody()
    {
        String phoneNum = "";
        BodyContent bodyFromTag = getBodyContent();
        String bodyText = bodyFromTag.getString();

        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        Connection con =
            DriverManager.getConnection("jdbc:odbc:mydatabase");
        Statement stmt = con.createStatement();
        String sqlQuery = "SELECT * FROM employees WHERE (name=' " +
            bodyText + "')";
        ResultSet rs = stmt.executeQuery(sqlQuery);
        rs.next();
        phoneNum = rs.getString(3);

        String outText = (bodyText + " is at extension " + phoneNum);
        bodyFromTag.getEnclosingWriter().print(outText);

        return SKIP_BODY;
    }
}
```

9 Type the code that creates a connection to the database.

10 Type the code that retrieves information from the database and stores it in a result set.

11 Type the code that retrieves the information from the result set and then type the code that uses the retrieved information.

12 Type the name of the `BodyContent` object followed by `.getEnclosingWriter().print()`. Between the second set of parentheses, type any arguments the method requires.

```
try
{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
}
catch(ClassNotFoundException e)
{
    /*Code to handle error*/
}

try
{
    Connection con =
        DriverManager.getConnection("jdbc:odbc:mydatabase");
    Statement stmt = con.createStatement();
    String sqlQuery = "SELECT * FROM employees WHERE (name=' " +
        bodyText + "')";
    ResultSet rs = stmt.executeQuery(sqlQuery);
    rs.next();
    phoneNum = rs.getString(3);
}
catch(SQLException e)
{
    /*Code to handle error*/
}

try
{
    String outText = (bodyText + " is at extension " + phoneNum);
    bodyFromTag.getEnclosingWriter().print(outText);
}
```

13 Create `try` and `catch` blocks that will handle any exceptions thrown while accessing the information from the database.

14 Save the file with the `.java` extension and then compile the source code for the file.

15 Copy the compiled class file to the appropriate directory on your Web server.

You can now add the tag handler class to a tag library descriptor file as you would for any custom tag with a body.