Java Server Pages

FANDS INFONET PVT.LTD.
1, Sheshadri, 1, Rambaug Colony, Paud Road, Pune 411 038.

Web - www.fandsindia.com
Email - fands@vsnl.com
Tel - 91-20-25463981 / 25468882

CHAPTER 1. Custom JSP Tags (actions)

1.1. Custom JSP Tags - What are they?

The idea behind creating JSP Tags is the ability to create new tags, storing them in specific file and then use them in different JSP Documents. In order to create new JSP Tags you need to create the Tag Handler class that will be connected to the new defined tag and a tag library descriptor file that connects between the created class to the new JSP Tag. Only after that, it is possible using the taglib directive, to use the new JSP Tag within a JSP Document.

1.2. A Simple Custom JSP Tag

When a new JSP Tag is created there is a need in declaring a new class, the handler class, that tells the JSP container what to do when it sees the new costumed JSP Tag.
The handler class must implement the javax.servlet.jsp.tagext.Tag interface. One of the options is declaring a class that extends the TagSupport or the BodyTagSupport classes.
Please notice the methods doStartTag() and doEndTag() that are called when arriving to the beginning tag and the ending tag respectively.
The following example presents a simple class declaration that can be used as a Tag Handler class that performs several actions whenever the new created JSP Tag is used.

```
package customt;

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

public class MyFirstTag extends TagSupport
{
        public int doStartTag()
        {
```

```
		try
		{
			pageContext.getOut().print("WE LOVE JAVA :)");
		}
		catch(IOException e)
		{
			e.printStackTrace();
		}
		return SKIP_BODY;
	}

	public int doEndTag()
	{
		try
		{
			pageContext.getOut().print("and JINI too !");
		}
		catch(IOException e)
		{
			e.printStackTrace();
		}
		return SKIP_PAGE;
	}
}
```

In order to make the logic connection between the Tag Handler class and the new Tag there is a need in a descriptive file to do it. The descriptive file has an xml format and its extension is *.tld. The descriptive file can be saved in the same directory in which the JSP Documents are saved.

The following descriptive file makes a logic connection between the MyFirstTag class to the new defined tag:jajalove.

```
<?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">

<!-- a tag library descriptor -->

<taglib>
  <!-- after this the default space is
         "http://java.sun.com/j2ee/dtds/jsptaglibrary_1_2.dtd"
```

```
 -->

 <tlibversion>1.0</tlibversion>
 <jspversion>1.1</jspversion>
 <shortname>custom</shortname>
 <uri></uri>
 <info>
        A simple tab library created by custom
 </info>

 <tag>
  <name>jajalove</name>
  <tagclass>customt.MyFirstTag</tagclass>
  <bodycontent>EMPTY</bodycontent>
  <info>display a java love message</info>
 </tag>

</taglib>
```

Now, in order to use the new defined tag in a JSP Document there is a need in using the taglib directive. The attribute uri denotes the url address of the descriptive file (it can be a full address or a part url address). The attribute prefix denotes the prefix that is used whenever one of the JSP Tags that the descriptive file describes is called. The following JSP Document presents a simple example of using the new JSP Tag.

```
<%@ page language="java" %>

<HTML>

        <HEAD>
                <TITLE>myFirstJspTag.jsp</TITLE>
        </HEAD>

        <BODY>
                <P><H1>myFirstJspTag.jsp</H1></P>

            <%@ taglib uri="custom-taglib.tld" prefix="custom" %>
              <custom:jajalove />

        </BODY>
<HTML>
```

In order to test the last example you should save the descriptive file it in the same directory in which the JSP Document is saved. More details regarding the meaning of each one of the elements within the descriptive file and the possibilities in declaring the Tag Handler class is presented later.

In creating a new simple JSP Tag (a Tag that describes what happens on behalf of a simple Tag that doesn't have attributes or a body) the associated class should extends the javax.servlet.jsp.tagext.TagSupport class. If the tag doesn't have attributes or body then override the doStartTag method, which defines the code that when the start tag is found is invoked. Note the use you can do in the pageContext, a predefined variable, that references an object on which you can invoke methods like getServletContext and getSession.

If the tag doesn't have a body then the doStartTag method should return SKIP_BODY constant.

1.3. Elements of the Tag Library Descriptor file

The best way in creating the descriptor file is simply taking a pre-made working one and make the needed changes. Usually those changes include the adding of Tag elements for each new defined tag.
Each tag element starting with <tag> and ending with </tag> has four necessary elements:

1. **name**
   This is the name of the new tag which will be written right after the suitable prefix.
2. **info**
   This is a short description of the new tag.
3. **tagclass**
   This is the fully qualified name of the class which is used as the Tag Handler.
4. **bodycontent**
   This Tag should have the value EMPTY when the new created Tag doesn't have one.

If the Tag has attributes, as will be explained later, then the attribute elements should be placed within the tag element as well. Each attribute in the new tag should have correspondent attribute element which has a starting tag and ending one: <attribute> and </attribute>. Within the attribute element you should add the following elements:

1. The name element to describe the attribute name
2. The required element has one of the two values: 'true', if giving the attribute a value is necessary in activating the tag or 'false' if giving the attribute a value isn't necessary.
3. The rtexpvalue element is optional and it has one of the two values: 'true' if the values given to the attribute must be static (meaning that they must be fixed) or 'false' if

the values given to the attribute can be determined at request time (using JSP Expression for instance). The default value is 'false'.

1.4. A Custom JSP Tag that has Attributes

In case of creating a pre-defined JSP Tag that has attributes, each attribute in the tag results in a call to the relative method which its name includes the attribute name (If there is an attribute with the name 'size' then a method named setSize will be called on the object that represents the JSP Tag). The value of the attribute will be passed to the appropriate method as a String. The common way of handling the attribute value, which is passed to the appropriate method, is storing it in a variable (usually an instance variable).

1.5. A custom JSP Tag that has a body

In order to create a custom JSP Tag that has a body the doStartTag method in the class that behinds the Tag should return EVAL_BODY_INCLUDE. By doing so, the body of the tag (which is what written between the starting tag and the ending one) will be displayed. Please notice that the bodycontent element in the descriptive file should have the value JSP.

1.6. A Custom JSP Tag that goes over its Body

In creating a JSP Tag that goes over the content of its body the defined class should extend the class BodyTagSupport instead of extending the class TagSupport. The class BodyTagSupport extends TagSupport.
The important methods the BodyTagSupport class has are:

1. **doAfterBody**
   This method should be overridden and return SKIP_BODY so the processing of the body will be stoped.
2. **getBodyContent**
   This method returns a BodyContent object that encapsulates information about the tag's body.
   The BodyContent has the following important methods:
   a. **getString**
      This method returns a String that contains the entire tag's body.
   b. **getEnclosingWriter**
      This method returns the JspWriter object that has been used by the doStartTag and the doEndTag
      methods.
   c. **getReader**

This method returns a Reader object that can be used to read the tag's body.

d. clearBody

This method clears the tag's body content.