To use a custom tag, use the following syntax:

The custom tag can be used in two ways. They are as follows:

<prefix:tag name attr1=value1.... attr = value />

<prefix:tag name attr1=value1.... attr = value>

   body code

</prefix:tagname>

We'll need three things to make a custom tag:

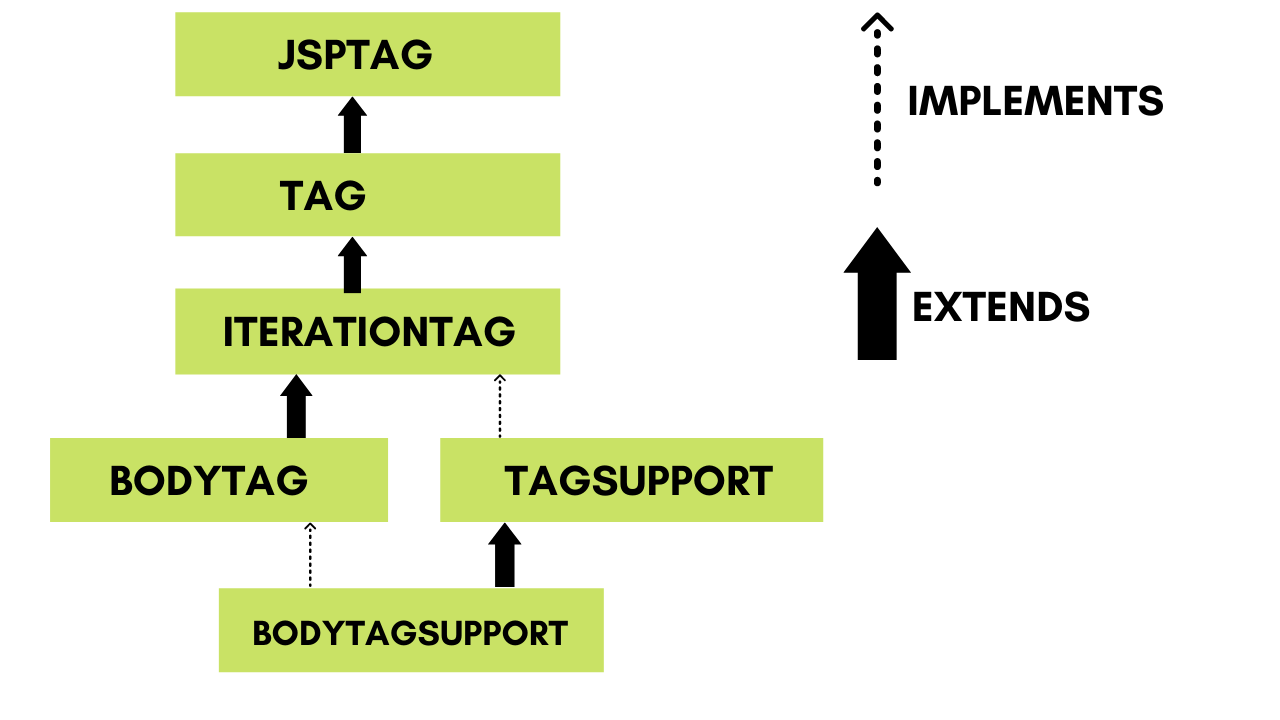
1. **Tag handler class:** We define what our custom tag will do when used in a JSP page in this class.

2.**Tag descriptor file (TLD):**Itcontains the tag name, tag handler class, and tag attributes.

3. **JSP page:** This is the JSP page where we'll use our custom tag.

**JSP Custom Tag API**

Classes and interfaces for the JSP custom tag API can be found in the javax.servlet.jsp.tagext package. In the Custom Tag hierarchy, the JspTag is the root interface.



**JspTag Interface**

JspTag is the root interface for all of the custom tag's interfaces and classes. It's a marker-based user interface.

**Tag Interface**

Jsp Tags Tag interface is a subinterface of JspTag. It includes ways for taking action at the beginning and conclusion of the tag.

**Tag interface fields**

In the Tag interface, there are four fields defined. They are as follows:

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| **public static int EVAL\_BODY\_INCLUDE** | It assesses the body's contents. |
| **public static int EVAL\_PAGE** | After the custom tag, it examines the JSP page content. |
| **public static int SKIP\_BODY** | It skips through the tag's body information. |
| **public static int SKIP\_PAGE** | After the custom tag, it skips the JSP page content. |

**Tag interface methods**

The Tag interface has the following methods:

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| **public void setPageContext(PageContext pc)** | It creates a PageContext object with the supplied parameters. |
| **public void setParent(Tag t)** | It determines the tag handler's parent. |
| **public Tag getParent()** | It returns the tag handler object for the parent tag. |
| **public int doStartTag()throws JspException** | The JSP page implementation object uses it to call it. The JSP programmer should override this function and define the business logic that will be executed at the beginning of the tag. |
| **public int doEndTag()throws JspException** | The JSP page implementation object uses it to call it. The JSP programmer should override this function and specify the business logic that will be executed at the tag's end. |
| **public void release()** | The JSP page implementation object calls it to release the state. |

**Advantages of Using Custom Tags**

The following are the main benefits of custom tags:

1. Removes the necessity for the scriptlet tag: The scriptlet tag is no longer required, which is considered a bad programming practice in JSP.

2. Business logic is separated from JSP: The custom tags keep the business logic separate from the JSP page, making it easier to maintain.

3. Adaptability: The custom tags allow you to reuse the same business logic repeatedly.

**JSP Custom Tag Handler**

This is the first step in using JSP to create custom tags. We inherit the TagSupport class and override the function doStartTag() to create the Tag Handler. The JspWriter class is used to write data to the JSP.

The getOut() method of the PageContext class returns an instance of the JspWriter class. By default, the TagSupport class gives a pageContext instance.

**ThisIsTagHandler.java**

package com.mycustomtag.sonoo;

import java.util.Calendar;

import javax.servlet.jsp.JspException;

import javax.servlet.jsp.JspWriter;

import javax.servlet.jsp.tagext.TagSupport;

public class MyTagHandler extends TagSupport{

public int doStartTag() throws JspException {

   JspWriter out=pageContext.getOut();//JspWriter's instance is returned.

   try{

    out.print(Calendar.getInstance().getTime());//JspWriter is used to print the date and time.

   }catch(Exception e){System.out.println(e);}

   return SKIP\_BODY;//The tag's body content will not be evaluated.

}

}

**Creating a TLD (Tag Library Descriptor) File in JSP**

Once the tag handler class is complete, we must create a TLD file in the WEB-INF directory, which will be loaded by the container when the application is deployed.

Tag and Tag Handler classes are described in the Tag Library Descriptor (TLD) file.

**ThisIsTags.tld**

<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE taglib

       PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"

   "http://java.sun.com/j2ee/dtd/web-jsptaglibrary\_1\_2.dtd">

<taglib>

<tlib-version>1.0</tlib-version>

<jsp-version>1.2</jsp-version>

<short-name>simple</short-name>

<uri>http://tomcat.apache.org/example-taglib</uri>

<tag>

<name>today</name>

<tag-class>com.mycustomtag.sonoo.ThisIsTagHandler</tag-class>

</tag>

</taglib>

The URI element should be noted; we'll need to define it in our deployment description file. It's also worth noting that the characteristics, format and number are essential. If body content is empty, the tag will have no content.

**Creating a page using JSP Custom Tag**

In our JSP file, let's use the tag. We are directly supplying the path of the TLD file here. However, it is preferable to use the URI name instead of the complete path to the TLD file.

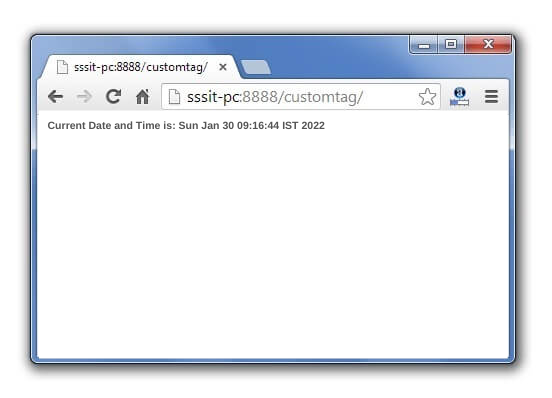
The tags defined in the TLD file are used using the taglib directive.

**index.jsp**

<%@ taglib uri="WEB-INF/mytags.tld" prefix="m" %>

Current Date and Time is: <m:today/>

After executing the code we get the following output:



**Custom Tag Attributes**

Along with your unique tags, you may use a variety of attributes. A custom tag class must implement the setter methods, which are the same as the JavaBean setter methods, in order to receive an attribute value.

package com.customtagmine;

import javax.servlet.jsp.tagext.\*;

import javax.servlet.jsp.\*;

import java.io.\*;

public class MyTag extends SimpleTagSupport {

  private String message;

  public void setMessage(String msg) {

     this.message = msg;

  }

  StringWriter sw = new StringWriter();

  public void doTag()

  throws JspException, IOException {

     if (message != null) {

        JspWriter out = getJspContext().getOut();

        out.println( message );

     } else {

        getJspBody().invoke(sw);

        getJspContext().getOut().println(sw.toString());

     }

  }

}

The setter method for the attribute "message" is setMessage (). Now, using the <attribute> element, let's add this attribute to the TLD file as follows:

<taglib>

  <tlib-version>1.0</tlib-version>

  <jsp-version>2.0</jsp-version>

  <short-name>TLD’s example with Body</short-name>

  <tag>

     <name>Hi</name>

     <tag-class>com.customtagmine.MyTag</tag-class>

     <body-content>scriptless</body-content>

     <attribute>

        <name>message</name>

     </attribute>

  </tag>

</taglib>

Let's have a look at JSP with the message attribute.

<%@ taglib prefix = "ex" uri = "WEB-INF/custom.tld"%>

<html>

  <head>

     <title>A example of custom tag</title>

  </head>

  <body>

     <ex:Hello message = "This is a example of custom tag" />

  </body>

</html>

As a result, you'll get the following outcome:

|  |
| --- |
| This is a example of custom tag |

Consider adding the properties below to an attribute:

|  |  |
| --- | --- |
| **S.No.** | **Property & Purpose** |
| **1** | **name**  An attribute's name is defined by the name element. For each tag, each attribute name must be unique. |
| **2** | **required**  This indicates whether this attribute is essential or optional. For optional, it would be false. |
| **3** | **rtexprvalue**  Declares whether a tag attribute's runtime expression value is legitimate. |
| **4** | **type**  This attribute's Java class-type is defined. It is considered to be String by default. |
| **5** | **description**  It is possible to provide an informative description. |
| **6** | **fragment**  Declares whether the value of this attribute should be regarded as a JspFragment. |

The following is an example of how to specify properties for an attribute.

.....

<attribute>

   <name>attr\_name</name>

   <required>false</required>

   <type>java.util.Date</type>

   <fragment>false</fragment>

</attribute>

.....

If you're utilising two attributes, you can change your TLD like this:

.....

<attribute>

   <name>attr\_my1</name>

   <required>false</required>

   <type>java.util.Boolean</type>

   <fragment>false</fragment>

</attribute>

<attribute>

   <name>attr\_my2</name>

   <required>true</required>

   <type>java.util.Date</type>

</attribute>

.....