* **Develop a jsp pages using library**

The JavaServer Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates the core functionality common to many JSP applications.

JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating the existing custom tags with the JSTL tags.

Install JSTL Library

To begin working with JSP tages you need to first install the JSTL library. If you are using the Apache Tomcat container, then follow these two steps −

**Step 1** − Download the binary distribution from [Apache Standard Taglib](https://tomcat.apache.org/taglibs/index.html) and unpack the compressed file.

**Step 2** − To use the Standard Taglib from its **Jakarta Taglibs distribution**, simply copy the JAR files in the distribution's 'lib' directory to your application's **webapps\ROOT\WEB-INF\lib** directory.

To use any of the libraries, you must include a <taglib> directive at the top of each JSP that uses the library.

Classification of The JSTL Tags

The JSTL tags can be classified, according to their functions, into the following JSTL tag library groups that can be used when creating a JSP page −

* **Core Tags**
* **Formatting tags**
* **SQL tags**
* **XML tags**
* **JSTL Functions**

Core Tags

The core group of tags are the most commonly used JSTL tags. Following is the syntax to include the JSTL Core library in your JSP −

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

Following table lists out the core JSTL Tags −

|  |  |
| --- | --- |
| **S.No.** | **Tag & Description** |
| 1 | [<c:out>](https://www.tutorialspoint.com/jsp/jstl_core_out_tag.htm)  Like <%= ... >, but for expressions. |
| 2 | [<c:set >](https://www.tutorialspoint.com/jsp/jstl_core_set_tag.htm)  Sets the result of an expression evaluation in a **'scope'** |
| 3 | [<c:remove >](https://www.tutorialspoint.com/jsp/jstl_core_remove_tag.htm)  Removes a **scoped variable** (from a particular scope, if specified). |
| 4 | [<c:catch>](https://www.tutorialspoint.com/jsp/jstl_core_catch_tag.htm)  Catches any **Throwable** that occurs in its body and optionally exposes it. |
| 5 | [<c:if>](https://www.tutorialspoint.com/jsp/jstl_core_if_tag.htm)  Simple conditional tag which evalutes its body if the supplied condition is true. |
| 6 | [<c:choose>](https://www.tutorialspoint.com/jsp/jstl_core_choose_tag.htm)  Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by **<when>** and **<otherwise>**. |
| 7 | [<c:when>](https://www.tutorialspoint.com/jsp/jstl_core_choose_tag.htm)  Subtag of **<choose>** that includes its body if its condition evalutes to **'true'**. |
| 8 | [<c:otherwise >](https://www.tutorialspoint.com/jsp/jstl_core_choose_tag.htm)  Subtag of **<choose>** that follows the **<when>** tags and runs only if all of the prior conditions evaluated to **'false'**. |
| 9 | [<c:import>](https://www.tutorialspoint.com/jsp/jstl_core_import_tag.htm)  Retrieves an absolute or relative URL and exposes its contents to either the page, a String in **'var'**, or a Reader in **'varReader'**. |
| 10 | [<c:forEach >](https://www.tutorialspoint.com/jsp/jstl_core_foreach_tag.htm)  The basic iteration tag, accepting many different collection types and supporting subsetting and other functionality . |
| 11 | [<c:forTokens>](https://www.tutorialspoint.com/jsp/jstl_core_foreach_tag.htm)  Iterates over tokens, separated by the supplied delimeters. |
| 12 | [<c:param>](https://www.tutorialspoint.com/jsp/jstl_core_param_tag.htm)  Adds a parameter to a containing **'import'** tag's URL. |
| 13 | [<c:redirect >](https://www.tutorialspoint.com/jsp/jstl_core_redirect_tag.htm)  Redirects to a new URL. |
| 14 | [<c:url>](https://www.tutorialspoint.com/jsp/jstl_core_url_tag.htm)  Creates a URL with optional query parameters |

**Formatting Tags**

The JSTL formatting tags are used to format and display text, the date, the time, and numbers for internationalized Websites. Following is the syntax to include Formatting library in your JSP −

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

Following table lists out the Formatting JSTL Tags −

|  |  |
| --- | --- |
| **S.No.** | **Tag & Description** |
| 1 | [<fmt:formatNumber>](https://www.tutorialspoint.com/jsp/jstl_format_formatnumber_tag.htm)  To render numerical value with specific precision or format. |
| 2 | [<fmt:parseNumber>](https://www.tutorialspoint.com/jsp/jstl_format_parsenumber_tag.htm)  Parses the string representation of a number, currency, or percentage. |
| 3 | [<fmt:formatDate>](https://www.tutorialspoint.com/jsp/jstl_format_formatdate_tag.htm)  Formats a date and/or time using the supplied styles and pattern. |
| 4 | [<fmt:parseDate>](https://www.tutorialspoint.com/jsp/jstl_format_parsedate_tag.htm)  Parses the string representation of a date and/or time |
| 5 | [<fmt:bundle>](https://www.tutorialspoint.com/jsp/jstl_format_bundle_tag.htm)  Loads a resource bundle to be used by its tag body. |
| 6 | [<fmt:setLocale>](https://www.tutorialspoint.com/jsp/jstl_format_setlocale_tag.htm)  Stores the given locale in the locale configuration variable. |
| 7 | [<fmt:setBundle>](https://www.tutorialspoint.com/jsp/jstl_format_setbundle_tag.htm)  Loads a resource bundle and stores it in the named scoped variable or the bundle configuration variable. |
| 8 | [<fmt:timeZone>](https://www.tutorialspoint.com/jsp/jstl_format_timezone_tag.htm)  Specifies the time zone for any time formatting or parsing actions nested in its body. |
| 9 | [<fmt:setTimeZone>](https://www.tutorialspoint.com/jsp/jstl_format_settimezone_tag.htm)  Stores the given time zone in the time zone configuration variable |
| 10 | [<fmt:message>](https://www.tutorialspoint.com/jsp/jstl_format_message_tag.htm)  Displays an internationalized message. |
| 11 | [<fmt:requestEncoding>](https://www.tutorialspoint.com/jsp/jstl_format_requestencoding_tag.htm)  Sets the request character encoding |

**SQL Tags**

The JSTL SQL tag library provides tags for interacting with relational databases (RDBMSs) such as **Oracle, mySQL**, or **Microsoft SQL Server**.

Following is the syntax to include JSTL SQL library in your JSP −

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

Following table lists out the SQL JSTL Tags −

|  |  |
| --- | --- |
| **S.No.** | **Tag & Description** |
| 1 | [<sql:setDataSource>](https://www.tutorialspoint.com/jsp/jstl_sql_setdatasource_tag.htm)  Creates a simple DataSource suitable only for prototyping |
| 2 | [<sql:query>](https://www.tutorialspoint.com/jsp/jstl_sql_query_tag.htm)  Executes the SQL query defined in its body or through the sql attribute. |
| 3 | [<sql:update>](https://www.tutorialspoint.com/jsp/jstl_sql_update_tag.htm)  Executes the SQL update defined in its body or through the sql attribute. |
| 4 | [<sql:param>](https://www.tutorialspoint.com/jsp/jstl_sql_param_tag.htm)  Sets a parameter in an SQL statement to the specified value. |
| 5 | [<sql:dateParam>](https://www.tutorialspoint.com/jsp/jstl_sql_dateparam_tag.htm)  Sets a parameter in an SQL statement to the specified java.util.Date value. |
| 6 | [<sql:transaction >](https://www.tutorialspoint.com/jsp/jstl_sql_transaction_tag.htm)  Provides nested database action elements with a shared Connection, set up to execute all statements as one transaction. |

**XML tags**

The JSTL XML tags provide a JSP-centric way of creating and manipulating the XML documents. Following is the syntax to include the JSTL XML library in your JSP.

The JSTL XML tag library has custom tags for interacting with the XML data. This includes parsing the XML, transforming the XML data, and the flow control based on the XPath expressions.

<%@ taglib prefix = "x"

uri = "http://java.sun.com/jsp/jstl/xml" %>

Before you proceed with the examples, you will need to copy the following two XML and XPath related libraries into your **<Tomcat Installation Directory>\lib** −

* **XercesImpl.jar** − Download it from <https://www.apache.org/dist/xerces/j/>
* **xalan.jar** − Download it from <https://xml.apache.org/xalan-j/index.html>

Following is the list of XML JSTL Tags −

|  |  |
| --- | --- |
| **S.No.** | **Tag & Description** |
| 1 | [<x:out>](https://www.tutorialspoint.com/jsp/jstl_xml_out_tag.htm)  Like <%= ... >, but for XPath expressions. |
| 2 | [<x:parse>](https://www.tutorialspoint.com/jsp/jstl_xml_parse_tag.htm)  Used to parse the XML data specified either via an attribute or in the tag body. |
| 3 | [<x:set >](https://www.tutorialspoint.com/jsp/jstl_xml_set_tag.htm)  Sets a variable to the value of an XPath expression. |
| 4 | [<x:if >](https://www.tutorialspoint.com/jsp/jstl_xml_if_tag.htm)  Evaluates a test XPath expression and if it is true, it processes its body. If the test condition is false, the body is ignored. |
| 5 | [<x:forEach>](https://www.tutorialspoint.com/jsp/jstl_xml_foreach_tag.htm)  To loop over nodes in an XML document. |
| 6 | [<x:choose>](https://www.tutorialspoint.com/jsp/jstl_xml_choose_tag.htm)  Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by **<when>** and **<otherwise>** tags. |
| 7 | [<x:when >](https://www.tutorialspoint.com/jsp/jstl_xml_choose_tag.htm)  Subtag of **<choose>** that includes its body if its expression evalutes to 'true'. |
| 8 | [<x:otherwise >](https://www.tutorialspoint.com/jsp/jstl_xml_choose_tag.htm)  Subtag of **<choose>** that follows the **<when>** tags and runs only if all of the prior conditions evaluates to 'false'. |
| 9 | [<x:transform >](https://www.tutorialspoint.com/jsp/jstl_xml_transform_tag.htm)  Applies an XSL transformation on a XML document |
| 10 | [<x:param >](https://www.tutorialspoint.com/jsp/jstl_xml_param_tag.htm)  Used along with the **transform** tag to set a parameter in the XSLT stylesheet |

**JSTL Functions**

JSTL includes a number of standard functions, most of which are common string manipulation functions. Following is the syntax to include JSTL Functions library in your JSP −

<%@ taglib prefix = "fn"

uri = "http://java.sun.com/jsp/jstl/functions" %>

Following table lists out the various JSTL Functions −

|  |  |
| --- | --- |
| **S.No.** | **Function & Description** |
| 1 | [fn:contains()](https://www.tutorialspoint.com/jsp/jstl_function_contains.htm)  Tests if an input string contains the specified substring. |
| 2 | [fn:containsIgnoreCase()](https://www.tutorialspoint.com/jsp/jstl_function_containsignorecase.htm)  Tests if an input string contains the specified substring in a case insensitive way. |
| 3 | [fn:endsWith()](https://www.tutorialspoint.com/jsp/jstl_function_endswith.htm)  Tests if an input string ends with the specified suffix. |
| 4 | [fn:escapeXml()](https://www.tutorialspoint.com/jsp/jstl_function_escapexml.htm)  Escapes characters that can be interpreted as XML markup. |
| 5 | [fn:indexOf()](https://www.tutorialspoint.com/jsp/jstl_function_indexof.htm)  Returns the index withing a string of the first occurrence of a specified substring. |
| 6 | [fn:join()](https://www.tutorialspoint.com/jsp/jstl_function_join.htm)  Joins all elements of an array into a string. |
| 7 | [fn:length()](https://www.tutorialspoint.com/jsp/jstl_function_length.htm)  Returns the number of items in a collection, or the number of characters in a string. |
| 8 | [fn:replace()](https://www.tutorialspoint.com/jsp/jstl_function_replace.htm)  Returns a string resulting from replacing in an input string all occurrences with a given string. |
| 9 | [fn:split()](https://www.tutorialspoint.com/jsp/jstl_function_split.htm)  Splits a string into an array of substrings. |
| 10 | [fn:startsWith()](https://www.tutorialspoint.com/jsp/jstl_function_startswith.htm)  Tests if an input string starts with the specified prefix. |
| 11 | [fn:substring()](https://www.tutorialspoint.com/jsp/jstl_function_substring.htm)  Returns a subset of a string. |
| 12 | [fn:substringAfter()](https://www.tutorialspoint.com/jsp/jstl_function_substringafter.htm)  Returns a subset of a string following a specific substring. |
| 13 | [fn:substringBefore()](https://www.tutorialspoint.com/jsp/jstl_function_substringbefore.htm)  Returns a subset of a string before a specific substring. |
| 14 | [fn:toLowerCase()](https://www.tutorialspoint.com/jsp/jstl_function_tolowercase.htm)  Converts all of the characters of a string to lower case. |
| 15 | [fn:toUpperCase()](https://www.tutorialspoint.com/jsp/jstl_function_touppercase.htm)  Converts all of the characters of a string to upper case. |
| 16 | [fn:trim()](https://www.tutorialspoint.com/jsp/jstl_function_trim.htm)  Removes white spaces from both ends of a string. |

* **DEVELOPING SIMPLE CUSTOM TAG**

A custom tag is a user-defined JSP language element. When a JSP page containing a custom tag is translated into a servlet, the tag is converted to operations on an object called a tag handler. The Web container then invokes those operations when the JSP page's servlet is executed.

JSP tag extensions lets you create new tags that you can insert directly into a JavaServer Page. The JSP 2.0 specification introduced the Simple Tag Handlers for writing these custom tags.

To write a custom tag, you can simply extend **SimpleTagSupport** class and override the **doTag()** method, where you can place your code to generate content for the tag.

Create "Hello" Tag

Consider you want to define a custom tag named <ex:Hello> and you want to use it in the following fashion without a body −

<ex:Hello />

To create a custom JSP tag, you must first create a Java class that acts as a tag handler. Let us now create the **HelloTag** class as follows −

package com.tutorialspoint;

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

import javax.servlet.jsp.\*;

import java.io.\*;

public class HelloTag extends SimpleTagSupport {

public void doTag() throws JspException, IOException {

JspWriter out = getJspContext().getOut();

out.println("Hello Custom Tag!");

}

}

The above code has simple coding where the **doTag()** method takes the current JspContext object using the **getJspContext()** method and uses it to send **"Hello Custom Tag!"** to the current **JspWriter** object

Let us compile the above class and copy it in a directory available in the environment variable CLASSPATH. Finally, create the following tag library file: **<Tomcat-Installation-Directory>webapps\ROOT\WEB-INF\custom.tld**.

<taglib>

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

<jsp-version>2.0</jsp-version>

<short-name>Example TLD</short-name>

<tag>

<name>Hello</name>

<tag-class>com.tutorialspoint.HelloTag</tag-class>

<body-content>empty</body-content>

</tag>

</taglib>

Let us now use the above defined custom tag **Hello** in our JSP program as follows −

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

<html>

<head>

<title>A sample custom tag</title>

</head>

<body>

<ex:Hello/>

</body>

</html>

Call the above JSP and this should produce the following result −

Hello Custom Tag!

Accessing the Tag Body

You can include a message in the body of the tag as you have seen with standard tags. Consider you want to define a custom tag named **<ex:Hello>** and you want to use it in the following fashion with a body −

<ex:Hello>

This is message body

</ex:Hello>

Let us make the following changes in the above tag code to process the body of the tag −

package com.tutorialspoint;

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

import javax.servlet.jsp.\*;

import java.io.\*;

public class HelloTag extends SimpleTagSupport {

StringWriter sw = new StringWriter();

public void doTag()

throws JspException, IOException {

getJspBody().invoke(sw);

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

}

}

Here, the output resulting from the invocation is first captured into a **StringWriter** before being written to the JspWriter associated with the tag. We need to change TLD file as follows −

<taglib>

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

<jsp-version>2.0</jsp-version>

<short-name>Example TLD with Body</short-name>

<tag>

<name>Hello</name>

<tag-class>com.tutorialspoint.HelloTag</tag-class>

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

</tag>

</taglib>

Let us now call the above tag with proper body as follows −

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

<html>

<head>

<title>A sample custom tag</title>

</head>

<body>

<ex:Hello>

This is message body

</ex:Hello>

</body>

</html>

You will receive the following result −

This is message body

Custom Tag Attributes

You can use various attributes along with your custom tags. To accept an attribute value, a custom tag class needs to implement the **setter** methods, identical to the JavaBean setter methods as shown below −

package com.tutorialspoint;

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

import javax.servlet.jsp.\*;

import java.io.\*;

public class HelloTag 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) {

/\* Use message from attribute \*/

JspWriter out = getJspContext().getOut();

out.println( message );

} else {

/\* use message from the body \*/

getJspBody().invoke(sw);

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

}

}

}

The attribute's name is **"message"**, so the setter method is **setMessage()**. Let us now add this attribute in the TLD file using the **<attribute>** element as follows −

<taglib>

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

<jsp-version>2.0</jsp-version>

<short-name>Example TLD with Body</short-name>

<tag>

<name>Hello</name>

<tag-class>com.tutorialspoint.HelloTag</tag-class>

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

<attribute>

<name>message</name>

</attribute>

</tag>

</taglib>

Let us follow JSP with message attribute as follows −

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

<html>

<head>

<title>A sample custom tag</title>

</head>

<body>

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

</body>

</html>

This will produce following result −

This is custom tag

Consider including the following properties for an attribute −

|  |  |
| --- | --- |
| **S.No.** | **Property & Purpose** |
| 1 | **name**  The name element defines the name of an attribute. Each attribute name must be unique for a particular tag. |
| 2 | **required**  This specifies if this attribute is required or is an optional one. It would be false for optional. |
| 3 | **rtexprvalue**  Declares if a runtime expression value for a tag attribute is valid |
| 4 | **type**  Defines the Java class-type of this attribute. By default it is assumed as **String** |
| 5 | **description**  Informational description can be provided. |
| 6 | **fragment**  Declares if this attribute value should be treated as a **JspFragment**. |

Following is the example to specify properties related to an attribute −

.....

<attribute>

<name>attribute\_name</name>

<required>false</required>

<type>java.util.Date</type>

<fragment>false</fragment>

</attribute>

.....

If you are using two attributes, then you can modify your TLD as follows −

.....

<attribute>

<name>attribute\_name1</name>

<required>false</required>

<type>java.util.Boolean</type>

<fragment>false</fragment>

</attribute>

<attribute>

<name>attribute\_name2</name>

<required>true</required>

<type>java.util.Date</type>

</attribute>

.....

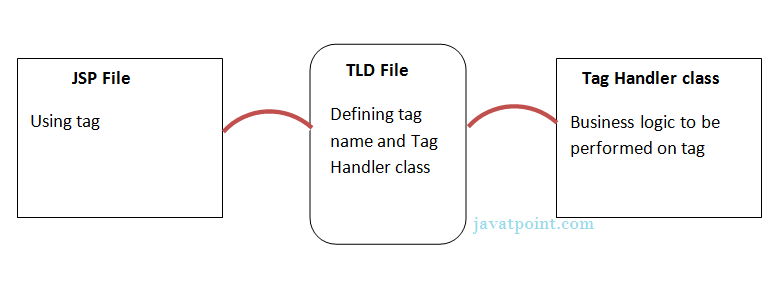
**Structure and Execution of a custom tag in jsp pages**

In this example, we are going to create a **custom tag that prints the current date and time**. We are performing action at the start of tag.

For creating any custom tag, we need to follow following steps:

1. **Create the Tag handler class** and perform action at the start or at the end of the tag.
2. **Create the Tag Library Descriptor (TLD) file** and define tags
3. **Create the JSP file that uses the Custom tag defined in the TLD file**

### **Understanding flow of custom tag in jsp**



### **1) Create the Tag handler class**

To create the Tag Handler, we are inheriting the **TagSupport class** and overriding its method **doStartTag()**.To write data for the jsp, we need to use the **JspWriter class**.

The **PageContext** class provides **getOut()** method that returns the instance of JspWriter class. TagSupport class provides instance of pageContext bydefault.

*File: MyTagHandler.java*

1. **package** com.javatpoint.sonoo;
2. **import** java.util.Calendar;
3. **import** javax.servlet.jsp.JspException;
4. **import** javax.servlet.jsp.JspWriter;
5. **import** javax.servlet.jsp.tagext.TagSupport;
6. **public** **class** MyTagHandler **extends** TagSupport{
8. **public** **int** doStartTag() **throws** JspException {
9. JspWriter out=pageContext.getOut();//returns the instance of JspWriter
10. **try**{
11. out.print(Calendar.getInstance().getTime());//printing date and time using JspWriter
12. }**catch**(Exception e){System.out.println(e);}
13. **return** SKIP\_BODY;//will not evaluate the body content of the tag
14. }
15. }

### **2) Create the TLD file**

**Tag Library Descriptor** (TLD) file contains information of tag and Tag Hander classes. It must be contained inside the **WEB-INF** directory.

*File: mytags.tld*

1. **<?xml** version="1.0" encoding="ISO-8859-1" **?>**
2. <!DOCTYPE taglib
3. PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
4. "http://java.sun.com/j2ee/dtd/web-jsptaglibrary\_1\_2.dtd"**>**
6. **<taglib>**
8. **<tlib-version>**1.0**</tlib-version>**
9. **<jsp-version>**1.2**</jsp-version>**
10. **<short-name>**simple**</short-name>**
11. **<uri>**http://tomcat.apache.org/example-taglib**</uri>**
13. **<tag>**
14. **<name>**today**</name>**
15. **<tag-class>**com.javatpoint.sonoo.MyTagHandler**</tag-class>**
16. **</tag>**
17. **</taglib>**

### **3) Create the JSP file**

Let's use the tag in our jsp file. Here, we are specifying the path of tld file directly. But it is recommended to use the uri name instead of full path of tld file. We will learn about uri later.

It uses **taglib** directive to use the tags defined in the tld file.

*File: index.jsp*

1. <%@ taglib uri="WEB-INF/mytags.tld" prefix="m" %>
2. Current Date and Time is: <m:today/>

[download this example](https://static.javatpoint.com/src/jsp/cu1.zip)

#### **Output**

