JSTL Quick Reference

Expressions (EL)

In Attribute Values

- `<a:tag value="${expr}"/>`
- `<a:tag value="me${expr} ${expr}"/>`

Bean Property Access

```
bean.name
bean["name"]
```

Indexed Property Access

```
bean.property[index]
```

Map Property Access

```
bean.property["key"]
```

Implicit Objects (maps)

Object	Description
pageContext	JSP Page Context object
pageScope	Page-scoped variables (valid only on a given JSP page)
requestScope	Request-scoped variables (valid for a given request)
sessionScope	Session-scoped variables (valid for the user's session)
applicationScope	Application-scoped variables (valid for a given application context)
param	Map of request parameter name to a String parameter value
paramValues	Map of request parameter name to a String array of parameter values
header	Map of request header name to a header String value
headerValues	Map of request header name to a String array of values
cookie	Map of cookie name to a Cookie object
initParam	Map of context initialization parameter name to a String parameter value (set in web.xml)

Arithmetic Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/ (div)	Division
% (mod)	Remainder (modulus)

Relational Operators

Operator	Description
== (eq)	Equality
!= (ne)	Inequality
< (lt)	Less than
> (gt)	Greater than
<= (le)	Less than or equal to
>= (ge)	Greater than or equal to
<= (le)	Less than or equal to

Logical Operators

Operator	Description
&& (and)	True if both operands are true; false, otherwise.
|| (or)	True if either or both operands are true; false, otherwise.
! (not)	True if operand is false; false, otherwise.

Other Operators

Operator	Description
empty	True if the operand is null, an empty String, empty array, empty Map, or empty List; false, otherwise.
()	Paranthesis for changing operator precedence.

EL Functions (*JSTL 1.1*)

Note: All functions treat null Strings as empty Strings.

```
<%@ taglib prefix="fn"
    uri="http://java.sun.com/jstl/functions" %>
```

Usage: ${fn:*function*(*arg0, ...*)}

```
<p>We offer ${fn:length(flavorSet)} ice cream
flavors.</p>
```

Function	Description
fn:contains (*string, substring*) : *boolean*	Returns true if *substring* is contained in *string*; false, otherwise.
fn:containsIgnoreCase (*string, substring*) : boolean	Returns true if *substring* is contained in *string* regardless of case; false, otherwise.
fn:endsWith (*string, suffix*) : boolean	Returns true if *string* ends with the specified *suffix*; false, otherwise.
fn:escapeXml (*string*) : String	Escapes characters (e.g changing “<” to “<”) that could be interpreted as XML (including HTML) markup.
fn:indexOf (*string, substring*) : int	Returns an integer representing the 0-based index within *string* of the first occurrence of *substring*. If *substring* is empty, 0 is returned.

Function	Description
fn:join (*string[], separator)* : String	Joins all elements of the *string* array into a single string. *Separator* separates each element in the resulting string. If *separator* is an empty string, the elements are joined without a separator.
fn:length (*collection* or *string*) : int	If a collection or array is passed, the size of the collection or array is returned; If a string is passed, the number of characters in the string is returned.
fn:replace (*inputString, beforeSubstring, afterSubstring)* : String	Replaces in *inputString*, every occurrence of *beforeString* with *afterString*. An empty string is returned if either *inputString* or *beforeString* is empty. If *afterString* is empty, all occurences of the *beforeString* are removed.
fn:split *(string, delimiters)* : String[]	Splits *string* into a string array using the given set of delimiter characters. The delimiter characters are not included in any returned tokens.
fn:startsWith *(string, prefix)* : boolean	Returns true if *string* starts with the specified *prefix*; false, otherwise. Returns true if *prefix* is empty.
fn:substring *(string, beginIndex, endIndex)* : String	Returns a subset of *string* using the zero-based indices – inclusive of the begin index, but exclusive of the end index.
fn:substringAfter *(string, substring)* : String	Returns the subset of *string* following the given *substring*.
fn:substringBefore *(string, substring)* : String	Returns the subset of *string* that procedes the given *substring*.
fn:toLowerCase *(string)* : String	Converts all characters of a string to lowercase.
fn:toUpperCase *(string)* : String	Converts all characters of a string to uppercase.
fn:trim *(string)* : String	Removes whitespace from both ends of a string.

Core Tag Library

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

General-Purpose Actions

Actions for rendering data, creating and modifying scoped variables, and catching exceptions.

<c:out> - renders data to the page

```
<h2>Welcome, <c:out value="${user.name}"
default="Guest"/></h2>
```

Attribute	Description	Rqd	Default
value	Data to output	Yes	*None*
default	Fallback data to output if *value* is empty	No	*Body*
escapeXml	`true` to escape special characters	No	true

<c:set> - saves data to a scoped variable

```
<c:set var="dogAge" value="${age div 7}"/>
You are <c:out value="${dogAge}"/> in dog
years.
```

Attribute	Description	Rqd	Default
value	Data to save	No	*Body*
target	Name of variable to modify	No	*None*
property	Property of target to modify	No	*None*
var	Name of variable to store data	No	*None*
scope	Scope of variable	No	page

<c:remove> - deletes a scoped variable

```
<c:remove var="dogAge" scope="page"/>
```

Attribute	Description	Rqd	Default
var	Name of variable to delete	Yes	*None*
scope	Scope of variable	No	*All scopes*

<c:catch> - traps all exceptions or errors from the enclosed body.

```
<c:catch var="err">
  <c:import value="http://java.sun.com"/>
</c:catch>
<c:if test="${not empty err}">
  Could not connect to Java web site.
</c:if>
```

Attribute	Description	Rqd	Default
var	Name of variable to hold the thrown exception, if any. Variable will be of type java.lang.Throwable.	No	*None*

Conditional Actions

Actions for processing markup based on logical conditions.

<c:if> - processes the body if *test* is true

```
<c:if test="${user.age ge 40}">
  You are over the hill.
</c:if>
```

Attribute	Description	Rqd	Default
test	Condition to evaluate	Yes	*None*
var	Name of variable to store test condition's result	No	*None*
scope	Scope of variable	No	page

<c:choose> - multiple conditions – processes the body of the **first** enclosed when tag where the test condition is true. If none match then the body of the otherwise tag (if present) is processed.

```
<c:choose>
```

```
  <c:when test="${a boolean expr}">
     // do something
  </c:when>
  <c:when test="${another boolean expr}">
     // do something else
  </c:when>
  <c:otherwise>
     // do this when nothing else is true
  </c:otherwise>
</c:choose>
```

The choose tag accepts no attributes and can only contain `when` *tag(s) and an optional* `otherwise` *tag.*

<c:when> - processes the body if *test* is true and no other previous `<c:when>` tags evaluated to true.

Attribute	Description	Rqd	Default
test	Condition to evaluate	Yes	*None*

<c:otherwise> - processes the body if no other previous `<c:when>` condition matched. This tag accepts no attributes and, if present, must be the last tag in the `<c:choose>` body.

Iterator Actions

Actions that loop over collections, for a fixed number of times, or over a set of string tokens. These actions share the following attributes for iterating over a subset of elements.

Attribute	Description	Rqd	Default
begin	Zero-based index of first item to process, inclusive.	No	0
end	Zero-based index of last item to process, inclusive.	No	*Last item*
step	Process every *step*th element (e.g 2 = every second element).	No	1
varStatus	Name of variable to hold the loop status with the following properties: • `index` – position of the current item • `count` – number of times through the loop (starting with 1) • `first` – boolean indicator if this is the first iteration • `last` – boolean indicator if this is the last iteration	No	*None*

<c:forEach> - repeats the nested body content over a collection or for a fixed number of times.

```
<c:forEach items="${user.languages}"
           var="lang" varStatus="status">
  <c:if test="${status.first}">
    You speak these languages:<br><ul>
  </c:if>
  <li><c:out value="${lang}"/></li>
  <c:if test="${status.last}"></ul></c:if>
</c:forEach>
```

Attribute	Description	Rqd	Default
var	Name of variable to hold the current item. This variable has only nested visibility.	No	*None*
items	Collection, iterator, map, or array to loop over.	No	*None*

<c:forTokens> - repeats the nested body content for each token of a delimited string.

```
<c:set var="users">Fred,Joe,Mary<c:set>
<c:forTokens var="name" items="${users}"
             delims=",">
  <c:out value="${name}"/><br/>
</c:forTokens>
```

Attribute	Description	Rqd	Default
var	Name of variable to hold the current token. This variable has only nested visibility.	No	*None*
items	String of tokens to loop over.	Yes	*None*
delims	Set of characters that separate the tokens (e.g. delims=",;" will tokenize a string separated by commas or semi-colons).	Yes	*None*

URL Related Actions

Actions for importing content from URLs, building URLs, and redirecting.

<c:import> - imports the content of a URL-based resource. Action may include nested `<c:param>` tags to specify the query string (unless the `varReader` attribute is specified).

```
<c:import url="includes/header.jsp">
  <c:param name="title">Hello World</c:param>
</c:import>
```

Attribute	Description	Rqd	Default
url	URL of the resource to import.	Yes	*None*
context	Name of the context (beginning with a /) of some other local web application to import the resource from.	No	*Current context*
var	Name of the variable to hold the imported content as a String.	No	*None*
scope	Scope of the *var* variable	No	page
varReader	Name of the variable to hold the imported content as a Reader. This variable has only nested visibility so that the reader will always be closed.	No	*None*

<c:url> - builds a URL with the proper rewriting rules applied (only relative URLs are rewritten). Action may include nested <c:param> tags to specify the query string.

```
<c:url="editProfile.do" var="profileLnk">
  <c:param name="id" value="${user.id}"/>
</c:url>
<a href='<c:out value="${profileLnk}"/>'>
  Edit Profile
</a>
```

Attribute	Description	Rqd	Default
value	URL to be processed.	Yes	*None*
context	Name of the context (beginning with a /) of some other local web application.	No	*Current context*
var	Name of the variable to hold the URL as a String.	No	*None*
scope	Scope of the *var* variable	No	page

<c:redirect> - sends the client a response to redirect to the specified URL. This action will abort processing of the current page. Action may include nested <c:param> tags to specify the query string.

```
<c:if test="${empty user}">
  <c:redirect url="login.do"/>
</c:if>
```

Attribute	Description	Rqd	Default
url	URL of the resource to redirect to.	Yes	*None*
context	Name of the context (beginning with a /) of some other local web application.	No	*Current context*

<c:param> - adds request parameters to a URL. This action can only be nested within <c:import>, <c:url>, or <c:redirect>.

Attribute	Description	Rqd	Default
name	Name of the query string parameter.	Yes	*None*
value	Value of the parameter. If not specified, value is taken from the tag body.	No	*Body*

Formatting Tag Library

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

Internationalization (I18N) Actions

Actions that establish localization (l10n) contexts, specify resource bundles, and format messages.

<fmt:setLocale> - Sets the default locale for the specified scope. This will override the browser-based locale.

```
<fmt:setLocale scope="session"
               value="fr_CA">
```

Attribute	Description	Rqd	Default
value	String representation of a locale (e.g. en_US) or an actual java.util.Locale object.	Yes	*None*
variant	Locale variant (as a String) to specify in conjunction with the locale (language and country).	No	*None*
scope	Scope to set the default locale for.	No	page

<fmt:bundle> - Sets the localization context, based on the specified resource bundle, to be used within the body content of this tag.

```
<fmt:bundle basename="resources"
            prefix="label.">
  <fmt:message key="userId"/>
<fmt:bundle>
```

Attribute	Description	Rqd	Default
basename	Fully-qualified name of the base bundle without a file type (such as ".properties").	Yes	*None*
prefix	String prefix to be prepended to the value of the message key. Note that the prefix must include *all* characters – a separator character (e.g. ".") is *not* assumed.	No	*None*

<fmt:setBundle> - Creates and stores in a scoped variable, a localization context based on the specified resource bundle.

```
<fmt:setBundle
     basename="ApplicationResources"
          var="strutsMessages"
        scope="application"/>
```

Attribute	Description	Rqd	Default
basename	Fully-qualified name of the base bundle without a file type (such as ".properties").	Yes	*None*
var	Name of the variable to hold the localization context.	No	*Default l10n context* (see "Configuration")
scope	Scope of the *var* variable	No	page

<fmt:message> - Looks up a localized message in a resource bundle. This tag can contain nested <fmt:param> tags to specify message format substitution values. The resultant message is printed or stored in a scoped variable.

```
<fmt:message key="title"
     bundle="${strutsResources}"/>
```

Attribute	Description	Rqd	Default
key	Message key to be looked up.	No	*Body*
bundle	Localization context (set by prio configuration, <fmt:bundle>, or <fmt:setBundle>, which specifies the resource bundle the message key is to be looked up in.	No	*Default l10n context*
var	Variable to hold the message.	No	
scope	Scope of the *var* variable.	No	page

<fmt:param> - Supplies a parameter for message format substitution in a containing `<fmt:message>` tag. Parameters are substituted in sequential order.

```
<fmt:message key="fieldRequired">
  <fmt:param value="User ID"/>
</fmt:message>
```

Attribute	Description	Rqd	Default
value	Value used for parametric message format substitution.	No	*Body*

<fmt:requestEncoding> - Instructs JSTL to use a specific character encoding (see http://www.iana.org/assignments/character-sets) to decode request parameters. Omitting a value indicates to use automatic detection of the proper encoding.

```
<fmt:requestEncoding key="ISO-8859-1"/>
```

Attribute	Description	Rqd	Default
value	Character encoding (e.g. "UTF-8") to use.	No	*Automatic*

Formatting Actions

Actions that format and parse numbers, currencies, percentages, dates and times.

<fmt:timeZone> - Sets the specified time zone to be applied to the nested body content. The following example demonstrates that the time zone by this action has only nested visibility.

```
<jsp:useBean id="now"
     class="java.util.Date"/>
<fmt:timeZone
     value="America/Los_Angeles">
  Pacific Time:<fmt:formatDate
  type="time" timeStyle="short"
  value="${now}"/>
</fmt:timeZone>
<br/>
Local Time:<fmt:formatDate type="time"
       timeStyle="short" value="${now}"/>
```

Attribute	Description	Rqd	Default
value	String representation of a time zone (such as "America/New_York", "GMT-5", or "EST"), or an actual java.util.TimeZone object.	Yes	*None*

<fmt:setTimeZone> - Sets the specified time zone in a named scoped variable or using the default time zone name if *var* is not specified.

```
<fmt:setTimeZone var="mtnTime"
     value="America/Denver"/>
Mountain Time: <fmt:formatDate
  type="time" timeStyle="short"
  value="${now}" timeZone="${mtnTime}"/>
```

Attribute	Description	Rqd	Default
value	String representation of a time zone (such as "America/New_York", "GMT-5", or "EST") or an actual java.util.TimeZone object.	Yes	*None*
var	Name of the variable to store the time zone.	No	*Default time zone* (see "Configuration")
scope	Scope of the *var* variable	No	page

<fmt:formatNumber> - Formats a number, currency, or percentage in a locale-sensitive manner. The formatted value is printed or stored in a scoped variable.

```
<fmt:formatNumber type="currency"
     value="3.977">
```

Attribute	Description	Rqd	Default
value	Numeric value to format.	No	*Body*
type	Specifies the type of value. Valid values are: • number • currency • percentage	No	number
pattern	Custom formatting pattern (overrides other formatting options including `type` – see java.text.DecimalFormat)	No	*None*

Attribute	Description	Rqd	Default
currencyCode	Currency code (ISO 4217) used for formatting currencies. Such as "USD" (US dollars) or "EUR" (euro).	No	*Based on default locale*
currencySymbol	Currency symbol used when formatting currencies. Such as "$" for US dollars, or "F" for Francs.	No	*Based on default locale*
groupingUsed	Specifies if grouping separators will be used (for example – formatting "23890" as "23,890").	No	true
maxIntegerDigits	Maximum number of digits to print in the integer part of the number.	No	*None*
minIntegerDigits	Minimum number of digits to print in the integer part of the number.	No	*None*
maxFractionDigits	Maximum number of digits to print in the fractional part of the number.	No	*None*
minFractionDigits	Minimum number of digits to print in the fractional part of the number.	No	*None*
var	Variable to store the formatted number.	No	*None*
scope	Scope of the *var* variable	No	page

<fmt:parseNumber> - Parses a String representing a number, currency, or percentage in a locale-sensitive manner. The parsed value is printed or stored in a scoped variable.

```
<fmt:parseNumber var="num" type="number"
     pattern="#,###" value="2,447"/>
<c:out value="${num}"/>
```

Attribute	Description	Rqd	Default
value	Value to parse.	No	*Body*
type	Specifies the type of value. Valid values are: • number • currency • percentage	No	number
pattern	Custom parsing pattern (overrides `type` – see java.text.DecimalFormat)	No	*None*
parseLocale	String representation of a locale (e.g. en_US) or an actual java.util.Locale object used for parsing.	No	*Default locale*
integerOnly	Specifies if only the integer portion of the value should be parsed.	No	false
var	Variable to store the formatted number.	No	*None*
scope	Scope of the *var* variable	No	page

Formatting Dates

Dates are formatted and parsed using the `<fmt:formatDate>` and `<fmt:parseDate>` actions which share the following common attributes.

Attribute	Description	Rqd	Default
type	Specifies the type of value. Valid values are: • time *(time only)* • date *(date only)* • both *(date and time)*	No	date
dateStyle	Predefined formatting style for a date (ignored if type="time") -- see java.text.DateFormat. Valid values are: • `default` (Jul 19, 2003) • `short` (7/19/03) • `medium` (Jul 19, 2003) • `long` (July 19, 2003) • `full` (Saturday, July 19, 2003)	No	default
timeStyle	Predefined formatting style for a time (ignored if type="date") -- see java.text.DateFormat. Valid values are: • `default` (2:51:16 PM) • `short` (2:51 PM) • `medium` (2:51:16 PM) • `long` (2:51:16 PM EDT) • `full` (2:51:16 PM EDT)	No	default
pattern	Custom formatting style (overrides `type, dateStyle,` and `timeStyle)` – see java.text.SimpleDateFormat.	No	*None*
timeZone	String representation of a time zone or an actual java.util.TimeZone object.	No	*Default time zone*

<fmt:formatDate> - Formats a date and/or time in a locale-sensitive manner. The formatted value is printed or stored in a scoped variable.

```
<fmt:formatDate value="${now}"
      pattern="yy-MMM-dd"/>
```

Attribute	Description	Rqd	Default
value	Date value to format. Value must be a java.util.Date object.	No	*None*
var	Variable to store the formatted number.	No	*None*
scope	Scope of the *var* variable	No	page

<fmt:parseDate> - Parses a string representing a date and/or time in a locale-sensitive manner. The parsed value is printed or stored in a scoped variable.

```
<fmt:parseDate var="bday"
     pattern="MM/dd/yy"
       value="05/10/63"/>
<fmt:formatDate value="${bday}"
     dateStyle="full"/>
```

Attribute	Description	Rqd	Default
value	Date/time string to parse.	No	*None*
parseLocale	String representation of a locale (e.g. en_US) or an actual java.util.Locale object used for parsing.	No	*Default locale*
var	Variable to store the parsed value as a java.util.Date.	No	*None*
scope	Scope of the *var* variable	No	page

SQL Tag Library

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

The SQL tag library provides actions to perform transactional database queries and updates and easily access query results.

<sql:query> - Performs a database query (e.g. *select* statement). The query should be expected to return a ResultSet. This action may include body content containing the actual query string as well as `<sql:param>` and `<sql:dateParam>` tags for parameter substitution. If the body contains the SQL query, it must appear before any nested parameter tags.

```
<sql:query var="users">
  SELECT * FROM users WHERE
    last_name = "Burdell"
</sql:query>
<c:forEach var="user"items="${users.rows}">
  <c:out value="${users.user_name}"/><br/>
</c:forEach>
```

Attribute	Description	Rqd	Default
sql	SQL query to execute.	No	*Body*
dataSource	Database connection to use. A javax.sql.DataSource object, or a String representing a JNDI resource or the parameters for java.sql.DriverManager. If specified, this action cannot be nested in <sql:transaction>.	No	*Default data source*
maxRows	Maximum number of rows to return in the result. If not specified or set to -1, no limit is enforced.	No	*Default maxRows* (see "Configuration")
startRow	Returned results include rows starting at this index. The first row is row 0.	No	0
var	Name of the variable to store the query results. Returned rows are accessible via the *rows* property of this object.	Yes	*None*
scope	Scope of the variable *var*.	No	page

<sql:update> - Performs a database insert, update, delete or other statement (such as a DDL statement) that does not return any results. This action may include body content containing the actual update string as well as `<sql:param>` tags for parameter substitution. Like `<sql:query>`, the `<sql:param>` tags must occur after the *sql* statement if it is contained in the `<sql:update>` tag body.

```
<sql:update var="updateCount">
  UPDATE users SET first_name="William" WHERE
    first_name = "Bill"
</sql:update>
<c:out value="${updateCount} rows updated."/>
```

Attribute	Description	Rqd	Default
sql	SQL statement to execute.	No	*Body*
dataSource	Database connection to use. A javax.sql.DataSource object, or a String representing a JNDI resource or the parameters for java.sql.DriverManager. If specified, this action cannot be nested in <sql:transaction>.	No	*Default data source*
var	Name of the variable to store the query results as a java.lang.Integer object (e.g. SQL *update* statements return the number of rows affected).	No	*None*
scope	Scope of the variable *var*.	No	page

<sql:transaction> - Establishes a database transaction for `<sql:query>` and `<sql:update>` tags contained in this tag's body. That is, all SQL actions contained in the body of this tag will be treated as a single atomic transaction. The transaction will be committed only if *all* statements succeed. If *any* of the contained SQL actions cause an exception, the transaction will be rolled back.

Note: this action will reinstate the prior "auto commit" setting after completion.

```
<sql:transaction>
  <sql:update sql="DELETE users WHERE
       user_name='bsiggelkow'"/>
  <sql:update sql="INSERT INTO users
   VALUES ('billsigg','Bill','Siggy')"/>
</sql:transaction>
```

Attribute	Description	Rqd	Default
dataSource	Database connection to use. A javax.sql.DataSource object, or a String representing a JNDI resource or the parameters for java.sql.DriverManager.	No	*Default data source* (see "Configuration")
isolation	Transaction isolation level. Valid values are: • `read_committed` • `read_uncommitted` • `repeatable_read` • `serializable`	No	*Database's default*

<sql:setDataSource> - Creates and stores in a scoped variable an SQL data source. This tag cannot have a body. Either the `dataSource` or `url` attribute must be specified.

```
<sql:setDataSource var="testDB"
     url="jdbc:mysql://localhost/test,
          com.mysql.jdbc.Driver"/>
<sql:query var="users"
    dataSource="${testDB}">
  SELECT * FROM users
</sql:query>
```

Attribute	Description	Rqd	Default
dataSource	Database connection to use or create. A javax.sql.DataSource object, or a String representing a JNDI resource or the parameters for java.sql.DriverManager.	No	*None*
driver	Name of the JDBC driver class.	No	*None*
url	URL for the JDBC database connection.	No	*None*
user	Database user's username	No	*None*
password	Database user's password	No	*None*
var	Name of the variable to hold the data source.	No	*Default data source name* (see "Configuration")
scope	Scope of the variable *var*.	No	page

<sql:param> - Substitutes parameters into SQL prepared statement placeholders ("?"). This action can only be nested within `<sql:query>` or `<sql:update>` tags. If the enclosing tag specifies the SQL in the body, any <sql:param> tags must appear after the SQL. Parameters are substituted in sequential order.

```
<c:set var="firstName">Bill</c:set>
<sql:query var="users">
  SELECT user_name, last_name
  FROM users WHERE first_name = ?
  <sql:param value="${firstName}"/>
</sql:query>
```

Attribute	Description	Rqd	Default
value	Value of the parameter. If not specified, value is taken from the tag body.	No	*Body*

<sql:dateParam> - Substitutes time, date, or timestamp parameters into SQL prepared statement placeholders. This action can only be nested within `<sql:query>` or `<sql:update>` tags. Parameters are substituted in sequential order.

```
<fmt:parseDate var="ww2End"
     pattern="yyyy-MM-dd"
     value="1945-09-02"/>
<sql:query var="postWarBabies">
     SELECT user_name FROM user_profile
       WHERE birth_date > ?
  <sql:dateParam value="${ww2End}"
       type="date"/>
</sql:query>
```

Attribute	Description	Rqd	Default
value	Value of the parameter. Must be a java.util.Date object.	Yes	*None*
type	Indicates how the database should interpret the value. Valid values are: • `date` • `time` • `timestamp`	No	timestamp

XML Tag Library

```
<%@ taglib prefix="x"
    uri="http://java.sun.com/jstl/xml" %>
```

The XML tag library supports parsing of XML documents, selection of XML fragments, conditional and iterative processing based on XML content, and XSLT transformations.

A common pattern for using the XML tags is as follows:

1. Use <x:parse> to parse XML into a scoped variable. The XML can come from the body literally, from the body via <c:import>, or from the value attribute which may refer to any XML source.
   `<x:parse var="varName" ...>`

2. Use the scoped variable from <x:parse> to specify the XML document to use in XPath expressions.
   `<x:out select="$varName/XPath expression"/>`

Using JSTL Data as XPath Variables

Scoped variables can be used in XPath expressions (*$implicitObject:variableName*) similar to how they are used in EL (*${implicitObject.variableName}*). If the implicit object is omitted, scopes will be searched in standard order. **Note that the "." and "[]" notations cannot be used for accessing bean properties.**

The following example demonstrates the above techniques.

```
<%-- Create the XML --%>
<x:parse var="doc">
  <users>
    <user id="997">
      <first-name>George</first-name>
      <last-name>Burdell</last-name>
    </user>
    <user id="998">
      <first-name>Joseph</first-name>
      <last-name>Blough</last-name>
    </user>
  </users>
</x:parse>

<%-- Define a variable holding user ID --%>
<c:set var="userId" value="${user.id}"
scope="page"/>

<%-- Find the user with  matching ID --%>
<x:set var="user"
select="$doc//users/user[@id=$pageScope:user
Id]"/>

<%-- Say Hi to the user --%>
Hi  <x:out select="$user/first-name"/> !
```

XPath Abbreviated Syntax

Abbr.	XPath Axis	Example
`//`	descendant	`$doc//first-name`
`.`	self	`$doc/users/user/.`
`..`	parent	`$doc//[first-name='George']/../last-name`
`@`	attribute	`$doc//user[@id="997"]/first-name`
(blank)	child	`$doc/users/user[last-name='Burdell']`
`*`	*(matches any namespace)*	`$doc//*[local-name()='users']/user`

Useful XPath Functions

Following are some useful XPath functions that can be used in XPath expressions. This list does not include all XPath functions – just those deemed particularly useful when using the JSTL XML tags.

Function	Description
ceiling *(number or object):* number	Returns the smallest integer greater than or equal to *number* or *object* (as converted as if by the number() function).
concat (*string1, string 2, ...*): string	Concatenates the arguments in order from left to right. Non-string arguments are converted to strings as if by the string() function.
contains *(string1, string2)*: boolean	Returns true if *string1* contains *string2*; false, otherwise. Non-string arguments are converted to strings as if by the string() function.
count (*node-set*): number	Returns the number of nodes in the node set.
floor *(number or object):* number	Returns the greatest integer less than or equal to *number* or *object* (as converted as if by the number() function).
last(): number	Returns the number of nodes in the context node.
local-name *(node-set):* string	Returns the part of the element name *after* the colon (:) when namespaces are used. If node set is not specified the function is applied to the context node.
name *(node-set):* string	Returns the qualified element name, that is, *prefix*:*local-name,* when namespaces are used. If node set is not specified the function is applied to the context node.
namespace-uri *(node-set):* string	Returns the uri of the namespace of the given node set. If node set is not specified the function is applied to the context node.
not (*boolean or object*): boolean	Returns the logical inversion of the supplied argument. That is, if the argument is true, false is returned; if argument is false, true is returned.
number *(object):* number	Converts *object* (or the current node if *object* is not specifed) to a number. *True* is converted to 1, and *false* to 0. If the number cannot be converted *NaN* is returned.
position(): number	Returns the position of the current node in the current context node set. The first position is 1.

Function	Description
round (*number* or *object*): number	Returns the closest integer to *number* or *object* (as converted as if by the number() function). If two integers are equally close, the value closer to positive infinity is chosen. (e.g. round(3.5) returns 4, round(-3.5) returns -3).
starts-with (*string1*,*string2*): boolean	Returns true if string1 starts with string2; false, otherwise.
string (*object*): string	Converts the *object*, or the current node if *object* is not specifed, to a string. Generally, this function will output the body of the elements (e.g. `string(<a>b</a>)` returns "`b`").
string-length (*object*): number	Number of characters in the string. If *object* is not a string it is first converted to a string as if by the string() function.
substring *(string, index, length)*: string	Returns a substring of *string* starting at *index* and continuing for *length* characters. **The first character is position 1, *not* position 0 as in Java and JavaScript.**
substring-after (*string1, string2*): string	Returns the substring in *string1* following the first of occurrence of *string2* in *string1*.
substring-before (*string1, string2*): string	Returns the substring in *string1* preceding the first of occurrence of *string2* in *string1*.
sum (*node-set*): number	Converts each node in the set to a number, as if by the number() function, adds up the numbers and returns the sum.

General XML Actions

Actions for parsing XML, outputting to the page, and selecting XML fragments. The examples that follow demonstrate use of the XML tags for processing Rich Site Summary (RSS) feeds. RSS has more or less the following format:

```
<?xml version="1.0"?>
<rss version="0.91">
  <channel>
    <title>feed title</title>
    <link>source url</link>
    <description>feed desc</description>
    <item>
      <title>article title</title>
      <link>article url</link>
      <description>article
desc</description>
    </item>
    <item>
      <title>article title</title>
      <link>article url</link>
      <description>article
desc</description>
    </item>
  </channel>
</rss>
```

<x:parse> - Parses XML content, provided by the *value* attribute or the tags body, into a scoped variable(s). This variable can then be used for subsequent processing by other XML tags.

```
<c:import var="rss"
url="http://servlet.java.sun.com/syndication
/rss_java_highlights-PARTNER-20.xml"/>
<x:parse var="news" xml="${rss}"/>
```

Attribute	Description	Rqd	Default
doc	*(JSTL 1.1)* Text of the document to a parse as a String or Reader.	No	*Body*
xml	*(JSTL 1.0 – deprecated in 1.1)* Same as *doc* attribute.	No	*Body*
systemId	URI of the original document, used for entity resolution.	No	*None*
filter	XML filter to apply. Value should be of type *org.xml.sax.XMLFilter*.	No	*None*
var	Name of the variable to store the results – may be an implementation-specific object.	Yes	*None*
scope	Scope of the variable *var*.	No	page
varDom	Name of the variable to store the result – stored as a DOM object.	Yes	*None*
scopeDom	Scope to store variable *varDOM* in.	No	page

<x:out> - Prints the result of the XPath expression as a string.

```
RSS Channel:
<x:out select="$news//channel/title"/>
```

Attribute	Description	Rqd	Default
select	XPath expression.	Yes	*None*
escapeXml	`true` to escape special characters	No	true

<x:set> - Saves the result of the *select* XPath expression to a scoped variable. Returned value may be a node set (XML fragment), boolean, string, or number.

```
<x:set select="$news//channel/item"
          var="newsItems"/>
```

Attribute	Description	Rqd	Default
select	XPath expression.	No	*None*
var	Name of variable to store results. The variable type is equal to whatever is returned by the *select* expression. • Boolean – java.lang.Boolean • Number – java.lang.Number • Node or node set – implementation-dependent type	No	*None*
scope	Scope of variable *var*.	No	page

Flow Control Actions

Actions for processing markup based on logical and iterative conditions.

<x:if> - Processes the body if *select* XPath evaluates to true (following the rules of the *boolean()* XPath function).

```
<x:if select=
```

```
"$news//item[contains(description,'Linux')]">
 Linux is in the news today!
</x:if>
```

Attribute	Description	Rqd	Default
select	Test condition as an XPath expression. Body content is processed if the condition is true.	Yes	*None*
var	Name of variable to store test condition's result	No	*None*
scope	Scope of variable	No	page

<x:choose> -- Processes the body of the **first** enclosed `<x:when>` tag where the select XPath expression evaluates to true. If none match then the body of the `<x:otherwise>` tag (if present) is processed.

```
<x:choose>
  <x:when select="$news//item">
     We've got news :)
  </x:when>
  <x:otherwise>
    No news today :(
  </x:otherwise>
</x:choose>
```

The choose tag accepts no attributes and can only contain `<x:when>` *tag(s) and an optional* `<x:otherwise>` *tag.*

<x:when> - Represents an alternative in an `<x:choose>` tag. Processes the body if the *select* expression evaluates to true and no other previous `<x:when>` tags in the `<x:choose>` matched.

Attribute	Description	Rqd	Default
select	Test condition as an XPath expression.	Yes	*None*

<x:otherwise> - Processes the body if no other previous `<x:when>` condition in the `<x:choose>` matched. This tag accepts no attributes and, if present, must be the last tag in `<x:choose>` body.

<x:forEach> - Repeats the nested body content over a *node set* determined by an XPath expression, setting the context node to each element in the iteration.

```
<x:forEach select="$news//item">
  <a href='<x:out select="link"/>'>
  <x:out select="title"/></a><br/>
</x:forEach>
```

Attribute	Description	Rqd	Default
select	Name of variable to hold the current item. This variable has only nested visibility.	No	*None*
var	Name of variable to hold the current item. This variable has only nested visibility.	No	*None*

Transformation Actions

JSTL provides an `<x:transform>` tag for performing XSLT transformations. The `<x:param>` tag can be nested in the `<x:transform>` tag to set a parameter that is used in the stylesheet.

<x:transform> - Conducts an XSLT transformation on source XML. The source XML is provided by the `doc` attribute or the body of the tag. The XSL stylesheet is specified by the `xslt` attribute. While in most cases, the stylesheet will be set up by back-end code – it is possible to define the stylesheet inline and make it available with `<c:set>` as in the following example:

```
<c:set var="xsl">
  <?xml version="1.0" encoding="UTF-8"?>
  <xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform
">
    <xsl:template match="item">
      <li>
        <xsl:value-of select="title"/>
      </li>
    </xsl:template>
    <xsl:template match="/">
      <ol>
       <xsl:apply-templates
            select="//item"/>
      </ol>
    </xsl:template>
  </xsl:stylesheet>
</c:set>

<x:transform xml="${news}"
             xslt="${xsl}"/>
```

Attribute	Description	Rqd	Default
doc	*(JSTL 1.1)* Source XML document for the transformation. Value can be a String, Reader, javax.xml.transform.Source, org.w3c.dom.Document, or any value exported <x:parse> or <x:set>. If the value comes from <x:set> it cannot be a partial document.	No	*Body*

Attribute	Description	Rqd	Default
xml	*(JSTL 1.0 – deprecated in 1.1)* Same as *doc* attribute.	No	*Body*
docSystemId	*(JSTL 1.1)* URI of the source XML to used to resolve entity references.	No	*None*
xmlSystemId	*(JSTL 1.0 – deprecated in 1.1)* Same as *docSystemId.*	No	*None*
xslt	XSLT stylesheet to use. The value must be a String, Reader or an javax.xml.transform.Source object.	Yes	*None*
xsltSystemId	URI of the stylesheet to use to resolve entity references.	No	*None*
var	Variable to hold the result as a org.w3c.doc.Document object.	No	*Non e*
scope	Scope of the variable *var.*		
result	javax.xml.transform.Result object that holds or processes the transformed XML.	No	*None*

<x:param> - Sets a transformation parameter that will be passed to stylesheet which declares parameters using the `<xsl:param>` tag. The `<x:param>` tag can only be nested within an `<x:transform>` tag. Any `<x:param>` tags must come after the XML body content of the `<x:transform>` tag, if present.

```
<x:transform xml="${news}"
             xslt="${searchXsl}"/>
  <x:param name="searchParam">
    Web Services
  </x:param>
</x:transform>
```

Attribute	Description	Rqd	Default
name	Name of the parameter as a String. This name must match the name in the corresponding <xsl:param> XSLT tag.	Yes	*None*
value	Value of the parameter. If not specified, value is taken from the tag body.	No	*Body*

JSTL Configuration

JSTL can utilize configuration parameters for setting such things as a default application resource bundle, time zone, and data source. These values can be established using (1) *setVariable*-like tags (e.g. <fmt:setLocale>, <sql:setDataSource>), (2) programmatically using the JSTL Config API or by (3) the web container itself via context initialization parameters set in the application's *web.xml* file.

1. **Set by a JSTL action –** this allows specification of scope by the *scope* attribute.

```
<fmt:setLocale value="en_US" scope="session"/>
```

2. **Set Programmatically –** allows specification of scope via the Config API.

```
import javax.servlet.jst.jstl.core.Config;
...
Config.set( session,
  Config.FMT_LOCALE,
  new java.util.Locale("en_US") );
```

3. **Set by Context Initialization Parameters –** specifies value used if setting not found in any of the standard scopes.

```
<web-app>
  ...
  <context-param>
    <param-name>
      javax.servlet.jsp.jstl.fmt.locale
    </param-name>
    <param-value>
      en_US
    </param-value>
  </context-param>
  ...
</web-app>
```

Locale
Specifies the default locale to be used by the *formatting* tags. This variable sets an *application-based* locale and has priority over any *browser-based* locales.

Variable Name	javax.servlet.jsp.jstl.fmt.locale
Java Constant	Config.FMT_LOCALE
Type	String (e.g. en_US) or java.util.Locale
Set by	<fmt:setLocale>, Config API, web.xml

Fallback Locale
Specifies the default locale to be used for finding resource bundle messages. This locale will be used if a message cannot be found using any of the preferred locales.

Variable Name	javax.servlet.jsp.jstl.fmt.fallbackLocale
Java Constant	Config.FMT_FALLBACK_LOCALE
Type	String (e.g. en_US) or java.util.Locale
Set by	Config API, web.xml

I18n Localization Context
Specifies the default resource bundle to use for localized messages. If a String is used (e.g. via the <fmt:setBundle> tag or a context-initialization parameter), the value refers to the name of the resource bundle (including any package name but excluding *.properties* or *.class*).

Variable Name	javax.servlet.jsp.jstl.fmt.localizationContext
Java Constant	Config.FMT_LOCALIZATION_CONTEXT
Type	String (e.g. resources.ApplicationResources) or javax.servlet.jsp.jstl.fmt.LocalizationContext
Set by	<fmt:setBundle>, Config API, web.xml

Time Zone
Specifies the default time zone to be used by the *formatting* tags.

Variable Name	javax.servlet.jsp.jstl.fmt.timeZone
Java Constant	Config.FMT_TIMEZONE
Type	String (e.g. “America/Los_Angeles”) or java.uti.TimeZone
Set by	<fmt:setTimeZone>, Config API, web.xml

Data Source
Specifies the data source to be accessed by the SQL tags. The value can be specified as a javax.sql.DataSource object, as a string representing a JNDI relatove path (e.g. “jdbc/myDatabase” if the absolute JNDI name is “java:comp/env/jdbc/myDatabase”), or a JDBC parameters string (*“url, driver, user, password”*) (e.g. “jdbc:mysql://localhost/,com.mysql.Driver”).

Variable Name	javax.servlet.jsp.jstl.sql.dataSource
Java Constant	Config.SQL_DATA_SOURCE
Type	String (JNDI relative path or JDBC parameters string) or javax.sql.DataSource
Set by	<fmt:setDataSource>, Config API, web.xml

Max Rows
Sets the maximum number of rows that can be returned by any <sql:query> tag. A value of -1 indicates no limit.

Variable Name	javax.servlet.jsp.jstl.sql.maxRows
Java Constant	Config.SQL_MAX_ROWS
Type	Integer
Set by	Config API, web.xml