A JSTL reference

A.1 Expression language syntax

Chapter 2 covers the JSTL expression language. Section A.1 serves as a concise summary.

A.1.1 Implicit objects

The JSTL expression `${data}` indicates the scoped variable named `data`. Additionally, the expression language supports the following implicit objects:

Implicit object	Contains
`pageScope`	Scoped variables from page scope
`requestScope`	Scoped variables from request scope
`sessionScope`	Scoped variables from session scope
`applicationScope`	Scoped variables from application scope
`param`	Request parameters as strings
`paramValues`	Request parameters as collections of strings
`header`	HTTP request headers as strings
`headerValues`	HTTP request headers as collections of strings
`initParam`	Context-initialization parameters
`cookie`	Cookie values
`pageContext`	The JSP `PageContext` object for the current page

For example, the expression `${param.username}` indicates the request parameter named `username`.

A.1.2 Operators

JSTL's operators help you work with data.

Property access

To retrieve properties from collections, the JSTL expression supports the following operators:

- The dot (`.`) operator retrieves a named property. The expression `${user.iq}` indicates the `iq` property of the scoped variable named `user`.
- The bracket (`[]`) operator lets you retrieve named or numbered properties:
  - The expression `${user["iq"]}` has the same meaning as `${user.iq}`.
  - The expression `${row[0]}` indicates the first item in the `row` collection.

Checking for emptiness

The `empty` operator determines whether a collection or string is empty or `null`. For instance, `${empty param.firstname}` will be true only if a request parameter named `firstname` is not present. JSTL expressions can also compare items directly against the keyword `null`, as in `${param.firstname == null}`, but this is an advanced use.

Comparing variables

The JSTL expression language supports comparisons using the following operators:

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

Arithmetic

JSTL expressions can conduct arithmetic using the following operators:

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

In addition, the – operator can precede a single number to reverse its sign: `${-30}`, `${-discount}`.

Boolean logic

The comparison operators produce boolean expressions, and JSTL expressions can also access boolean primitives. To combine boolean subexpressions, JSTL provides the following operators:

Operator	Description
`&&` `and`	`True` only if both sides are true
`||` `or`	`True` if either or both sides are true
`!` `not`	`True` only if the expression following it is false

JSTL supports two boolean literals: `true` and `false`.

Parentheses

JSTL expressions can use parentheses to group subexpressions. For example, `${(1 + 2) * 3}` equals `9`, but `${1 + (2 * 3)}` equals `7`.

A.2 Core tag library

JSTL's core tag library supports output, management of variables, conditional logic, loops, text imports, and URL manipulation. JSP pages can import the core tag library with the following directive:

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

A.2.1 General-purpose tags

JSTL provides `<c:out>` for writing data, `<c:set>` for saving data to memory, `<c:remove>` for deleting data, and `<c:catch>` for handling errors.

Examples

```
Thanks for logging in, <c:out value="${name}"/>.
<c:set var="loggedIn" scope="session" value="${true}"/>
<c:remove var="loggedOut" scope="session"/>
```

Tag attributes

The `<c:catch>` tag's attribute is as follows:

Attribute	Description	Required	Default
`var`	Variable to expose information about error	No	*None*

The `<c:out>` tag's attributes are as follows:

Attribute	Description	Required	Default
`value`	Information to output	Yes	*None*
`default`	Fallback information to output	No	*Body*
`escapeXml`	`True` if the tag should escape special XML characters	No	`true`

The `<c:set>` tag's attributes are as follows:

Attribute	Description	Required	Default
`value`	Information to save	No	*Body*
`target`	Name of the variable whose property should be modified	No	*None*
`property`	Property to modify	No	`None`
`var`	Name of the variable to store information	No	*None*
`scope`	Scope of variable to store information	No	`page`

If `target` is specified, `property` must also be specified.

The `<c:remove>` tag's attributes are as follows:

Attribute	Description	Required	Default
`var`	Name of the variable to remove	Yes	*None*
`scope`	Scope of the variable to remove	No	*All scopes*

A.2.2 Conditional logic

JSTL has four tags for conditions: `<c:if>`, `<c:choose>`, `<c:when>`, and `<c:otherwise>`.

Examples

```
<c:if test="${user.wealthy}">
  You have quite a lot of money in your account.
</c:if>
```

```
<c:choose>
  <c:when test="${user.generous}">
    Why don't you give some of it to me?
  </c:when>
  <c:when test="${user.stingy}">
    You should transfer some of it to a CD.
  </c:when>
```

```
  <c:otherwise>
     A money-market account looks right for you.
  </c:otherwise>
</c:choose>
```

Tag attributes

The `<c:if>` tag's attributes are as follows:

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

The `<c:choose>` tag accepts no attributes.

The `<c:when>` tag's attribute is as follows:

Attribute	Description	Required	Default
`test`	Condition to evaluate	Yes	*None*

The `<c:otherwise>` tag accepts no attributes.

A.2.3 Looping

The core JSTL library offers two tags for looping: `<c:forEach>` for general data and `<c:forTokens>` for parts of strings.

Examples

```
<c:forEach items="${orders}" var="order">
  <c:out value="${order.id}"/>
</c:forEach>
```

```
<c:forEach begin="0" end="100">
  I will not continue to disrupt class discussions!
</c:forEach>
```

```
<c:forTokens items="a:b:c:d" delims=":" var="token">
  <c:out value="${token}"/>
</c:forTokens>
```

Tag attributes

The `<c:forEach>` tag's attributes are as follows:

Attribute	Description	Required	Default
`items`	Information to loop over	No	*None*
`begin`	Element to start with (`0` = first item, `1` = second item, ...)	No	`0`
`end`	Element to end with (`0` = first item, `1` = second item, ...)	No	*Last item in the collection*
`step`	Process every `step` items	No	`1` *(all items)*
`var`	Name of the variable to expose the current item	No	*None*
`varStatus`	Name of the variable to expose the loop status	No	*None*

Either `items`, or both `begin` and `end`, must be specified.

The `<c:forTokens>` tag's attributes are as follows:

Attribute	Description	Required	Default
`items`	String to tokenize	Yes	*None*
`delims`	Characters to use as delimiters	Yes	*None*
`begin`	Element to start with (`0` = first item, `1` = second item, ...)	No	`0`
`end`	Element to end with (`0` = first item, `1` = second item, ...)	No	*Last item in the collection*
`step`	Process every `step` items	No	`1` *(all items)*
`var`	Name of the variable to expose the current item	No	*None*
`varStatus`	Name of the variable to expose the loop status	No	*None*

A.2.4 Import and URL

The core library supports inclusion of text using `<c:import>`, URL printing and formatting with `<c:url>`, and redirections with `<c:redirect>`. All URL tags accept `<c:param>` child tags.

Examples

```
<c:import url="http://www.cnn.com/cnn.rss" var="newsfeed"/>

<a href="<c:url url="/index.jsp"/>"/>
```

```
<c:redirect url="go-away.jsp">
  <c:param name="unwantedUser" value="true"/>
</c:redirect>
```

Tag attributes

The `<c:import>` tag's attributes are as follows:

Attribute	Description	Required	Default
`url`	URL to retrieve and import into the page	Yes	*None*
`context`	/ followed by the name of a local web application	No	*Current application*
`charEncoding`	Character set to use for imported data (if necessary)	No	*ISO-8859-1*
`var`	Name of the variable to expose imported text	No	*Print to page*
`scope`	Scope of the variable to expose imported text	No	`page`
`varReader`	Name of an alternate variable to expose `java.io.Reader`	No	*None*

The `<c:url>` tag's attributes are as follows:

Attribute	Description	Required	Default
`value`	Base URL	Yes	*None*
`context`	/ followed by the name of a local web application	No	*Current application*
`var`	Name of the variable to expose the processed URL	No	*Print to page*
`scope`	Scope of the variable to expose the processed URL	No	`page`

The `<c:redirect>` tag's attributes are as follows:

Attribute	Description	Required	Default
`url`	URL to redirect the user's browser to	Yes	*None*
`context`	/ followed by the name of a local web application	No	*Current application*

The `<c:param>` tag's attributes are as follows:

Attribute	Description	Required	Default
`name`	Name of the request parameter to set in the URL	Yes	*None*
`value`	Value of the request parameter to set in the URL	No	*Body*

A.3 XML tag library

JSTL's XML-processing tag library supports parsing of XML documents, selection of XML fragments, flow control based on XML, and XSLT transformations. JSP pages can import the XML-processing tag library with the following directive:

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

A.3.1 Parsing and general manipulation

Before you work with an XML document, it must be parsed with `<x:parse>` or back-end Java code. The `<x:out>` and `<x:set>` tags can retrieve fragments of parsed documents, whether these documents are DOM objects or a JSTL implementation's own choice of data type.

Examples

```
<c:set var="textDocument">
  <orders>
    <order>
      762 cans of low-fat yogurt
    </order>
    <order>
      6 spoons
    </order>
  </orders>
</c:set>
<x:parse xml="${textDocument}" var="xml"/>
<x:out select="$xml/orders/order[1]"/>

<x:set var="fragment" select="$xml//order"/>
```

Tag attributes

The `<x:parse>` tag's attributes are as follows:

Attribute	Description	Required	Default
`xml`	Text of the document to parse (`String` or `Reader`)	No	*Body*
`systemId`	URI of the original document (for entity resolution)	No	*None*
`filter`	`XMLFilter` object to filter the document	No	*None*
`var`	Name of the variable to expose the parsed document	No	*None*
`scope`	Scoped of the variable to expose the parsed document	No	*None*
`varDom`	Name of the variable to expose the parsed DOM	No	*None*
`scopeDom`	Scoped of the variable to expose the parsed DOM	No	*None*

The `<x:out>` tag's attributes are as follows:

Attribute	Description	Required	Default
`select`	XPath expression to evaluate as a string, often using XPath variables	Yes	*None*
`escapeXml`	True if the tag should escape special XML characters	No	`true`

The `<x:set>` tag's attributes are as follows:

Attribute	Description	Required	Default
`select`	Any XPath expression, often using XPath variables	Yes	*None*
`var`	Name of the variable to store the XPath expression's result	Yes	*None*
`scope`	Scope of the variable to store the XPath expression's result	No	`page`

If the XPath expression results in a boolean, `<x:set>` exposes a `java.lang.Boolean` object; for a string, `java.lang.String`; and for a number, `java.lang.Number`. XPath node-sets are exposed using an implementation-dependent type.

A.3.2 Conditional logic

Like the core library, the XML library supports four tags for conditional logic: `<x:if>`, `<x:choose>`, `<x:when>`, and `<x:otherwise>`.

Examples

```
<c:set var="textDocument">
  <orders>
    <order>
      17 carts
    </order>
    <order>
      34 horses
    </order>
  </orders>
</c:set>
<x:parse xml="${textDocument}" var="xml"/>
<x:if select="$xml//order">
  Document has at least one &lt;order&gt; element.
</x:if>

<x:choose>
  <x:when test="$xml//order[1] = '17 carts'">
    Looks like you put the carts before the horses.
  </x:when>
  <x:when test="$xml//order[1] = '49 pigs'">
```

```
    Why do you need 49 pigs?
  </x:when>
  <x:otherwise>
    I don't know <i>what</i> you ordered.
    Buy some more stuff and give us another chance
    to figure it out.
  </x:otherwise>
</x:choose>
```

Tag attributes

The `<x:if>` tag's attributes are as follows:

Attribute	Description	Required	Default
`select`	XPath expression to evaluate as boolean, often using XPath variables	Yes	*None*
`var`	Name of the variable to store the condition's result	No	*None*
`scope`	Scope of the variable to store the condition's result	No	`page`

The `<x:choose>` tag accepts no attributes.

The `<x:when>` tag's attribute is as follows:

Attribute	Description	Required	Default
`select`	Condition to evaluate	Yes	*None*

The `<x:otherwise>` tag accepts no attributes.

A.3.3 Loops

Like the core library, the XML-processing library supports looping over data. It provides a single tag, `<x:forEach>`, to loop over nodes in an XML document.

Examples

```
<c:set var="textOrders">
  <orders>
    <order>
      12 gallons of strawberry margarita mix
    </order>
    <order>
      6 tons of pickled sausage
    </order>
    <order>
      17 mice
    </order>
  </orders>
```

```
</c:set>
<x:parse xml="${textOrders}" var="orders"/>

You ordered:
<ul>
<x:forEach select="$orders/orders/order" var="item">
  <li><x:out select="." /></li>
</x:forEach>
</ul>
```

Tag attributes

The `<x:forEach>` tag's attributes are as follows:

Attribute	Description	Required	Default
`select`	XPath expression pointing to a set of nodes (often using XPath variables)	Yes	*None*
`var`	Name of the variable to store the current item for each loop	No	*None*

A.3.4 Transformations

JSTL provides an `<x:transform>` tag for conducting XSLT transformations from within a JSP page.

The `<x:param>` tag can set a parameter in an XSLT stylesheet.

Examples

```
<x:transform xml='${xml}' xslt='${xslt}'/>

<x:transform xslt='${xslt}'>
  <orders>
    <order>
      16 boxes of dried cheese
    </order>
    <order>
      34 live cattle
    </order>
  </orders>
</x:transform>
```

Tag attributes

The `<x:transform>` tag's attributes are as follows:

Attribute	Description	Required	Default
`xml`	Source XML document for the XSLT transformation	No	*Body*
`xmlSystemId`	URI of the original XML document (for entity resolution)	No	*None*
`xslt`	XSLT stylesheet providing transformation instructions	Yes	*None*

Attribute	Description	Required	Default
`xsltSystemId`	URI of the original XSLT document (for entity resolution and XSLT tags like `<xsl:include>`)	No	*None*
`result`	`javax.xml.transform.Result` object to accept the transformation's result	No	*Print to page*
`var`	Name of the variable to expose the transformation's result	No	*Print to page*
`scope`	Scope of the variable to expose the transformation's result	No	`page`

The `<x:param>` tag's attributes are as follows:

Attribute	Description	Required	Default
`name`	Name of the XSLT parameter to set	Yes	*None*
`value`	Value of the XSLT parameter to set	No	*Body*

A.4 Database tag library

JSTL's database library supports database queries, updates, and transactions. JSP pages can import this library with the following directive:

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

A.4.1 Preparing databases

For JSP pages that do not have a default database, `<sql:setDataSource>` can prepare a database for use.

Examples

```
<sql:setDataSource
  driver="org.hsqldb.jdbcDriver"
  url="jdbc:hsqldb:/home/databases/orders"
  user="sa"
  password="shhhh!"/>
```

Tag attributes

The `<sql:setDataSource>` tag's attributes are as follows:

Attribute	Description	Required	Default
`driver`	Name of the JDBC driver class to be registered	No	*None*
`url`	JDBC URL for the database connection	No	*None*

Attribute	Description	Required	Default
`user`	Database username	No	*None*
`password`	Database password	No	*None*
`dataSource`	Database prepared in advance (`String` or `javax.sql.DataSource`)	No	*None*
`var`	Name of the variable to represent the database	No	*Set default*
`scope`	Scope of the variable to represent the database	No	`page`

A.4.2 Queries and updates

JSTL can read from databases with `<sql:query>` and write to them with `<sql:update>`. These tags support SQL commands with `?` placeholders, which `<sql:param>` and `<sql:dateParam>` can fill in.

Examples

```
<sql:query var="result">
  SELECT ORDER
  FROM ORDERS
  WHERE CUSTOMER_ID='52'
    AND PRODUCT_NAME='Oat Bran'
</sql:query>
<c:forEach items="${result.rows}" var="row">
  <c:out value="${row.product_name}"/>
</c:forEach>

<sql:update var="count">
  UPDATE CONVICTS
    SET ARRESTS=ARRESTS+1
    WHERE CONVICT_ID=?
  <sql:param value="${currentConvict}"/>
</sql:update>
```

Tag attributes

The `<sql:query>` tag's attributes are as follows:

Attribute	Description	Required	Default
`sql`	SQL command to execute (should return a `ResultSet`)	No	*Body*
`dataSource`	Database connection to use (overrides the default)	No	*Default database*
`maxRows`	Maximum number of results to store in the variable	No	*Unlimited*
`startRow`	Number of the row in the result at which to start recording	No	`0`

Attribute	Description	Required	Default
`var`	Name of variable to expose the result from the database	Yes	*None*
`scope`	Scope of variable to expose the result from the database	No	`page`

The `<sql:update>` tag's attributes are as follows:

Attribute	Description	Required	Default
`sql`	SQL command to execute (should not return a `ResultSet`)	No	*Body*
`dataSource`	Database connection to use (overrides the default)	No	*Default database*
`var`	Name of the variable to store the count of affected rows	No	*None*
`scope`	Scope of the variable to store the count of affected rows	No	`page`

The `<sql:param>` tag's attribute is as follows:

Attribute	Description	Required	Default
`value`	Value of the parameter to set	No	*Body*

The `<sql:dateParam>` tag's attributes are as follows:

Attribute	Description	Required	Default
`value`	Value of the date parameter to set (`java.util.Date`)	Yes	*None*
`type`	`DATE`[a] (date only), `TIME` (time only), or `TIMESTAMP` (date and time)	No	`TIMESTAMP`

a. In this appendix's tables, I list discrete sets of permissible values in ALL CAPS to help them stand out. JSTL is case-insensitive in this regard, and I encourage lowercase in practice because it makes the tags easier and more pleasant to read.

A.4.3 Transactions

JSTL provides an `<sql:transaction>` tag to group `<sql:query>` and `<sql:update>` into transactions.

Examples

```
<sql:transaction>
  <sql:update>
```

```
    UPDATE BALANCES
      SET BALANCE = BALANCE + 2
      WHERE USER=25
  </sql:update>
  <sql:update>
    UPDATE BALANCES
      SET BALANCE = BALANCE - 2
      WHERE USER=30
  </sql:update>
</sql:transaction>
```

Tag attributes

The `<sql:transaction>` tag's attributes are as follows:

Attribute	Description	Required	Default
`dataSource`	Database connection to use (overrides the default)	No	Default database
`isolation`	Transaction isolation (`READ_COMMITTED`, `READ_UNCOMMITTED`, `REPEATABLE_READ`, or `SERIALIZABLE`)	No	Database's default

A.5 Formatting tag library

JSTL's internationalization-capable formatting library supports localized formatting, fine-grained control over the display of numbers and dates, and internationalization of text messages. JSP pages can import this library with the following directive:

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

A.5.1 Numbers

JSTL provides the `<fmt:formatNumber>` tag to display numbers and the `<fmt:parseNumber>` tag to read numbers.

Examples

```
<fmt:formatNumber value="${balance}" type="currency"/>
<fmt:parseNumber value="${param.number}" var="numb"/>
```

Tag attributes

The `<fmt:formatNumber>` tag's attributes are as follows:

Attribute	Description	Required	Default
`value`	Numeric value to display	No	*Body*
`type`	`NUMBER`, `CURRENCY`, or `PERCENT`	No	`number`
`pattern`	Custom formatting pattern	No	*None*
`currencyCode`	Currency code (for `type="currency"`)	No	*From the default locale*
`currencySymbol`	Currency symbol (for `type="currency"`)	No	*From the default locale*
`groupingUsed`	Whether to group numbers (`TRUE` or `FALSE`)	No	`true`
`maxIntegerDigits`	Maximum number of integer digits to print	No	*None*
`minIntegerDigits`	Minimum number of integer digits to print	No	*None*
`maxFractionDigits`	Maximum number of fractional digits to print	No	*None*
`minFractionDigits`	Minimum number of fractional digits to print	No	*None*
`var`	Name of the variable to store the formatted number (as a text string)	No	*Print to page*
`scope`	Scope of the variable to store the formatted number	No	`page`

The `<fmt:parseNumber>` tag's attributes are as follows:

Attribute	Description	Required	Default
`value`	Numeric value to read (parse)	No	*Body*
`type`	`NUMBER`, `CURRENCY`, or `PERCENT`	No	`number`
`parseLocale`	Locale to use when parsing the number	No	*Default locale*
`integerOnly`	Whether to parse to an integer (`true`) or floating-point number (`false`)	No	`false`
`pattern`	Custom parsing pattern	No	*None*
`var`	Name of the variable to store the parsed number (as a text string)	No	*Print to page*
`scope`	Scope of the variable to store the parsed number	No	`page`

A.5.2 Dates

To read and write dates, JSTL provides `<fmt:parseDate>` and `<fmt:formatDate>`, respectively. To adjust the time zones used for reading and writing dates, JSTL offers the `<fmt:timeZone>` and `<fmt:setTimeZone>` tags.

Examples

```
<fmt:formatDate value="${birthday}"/>
  <fmt:parseDate value="${birthday}" var="date"/>

<sql:query>
  …
    <sql:dateParam value="${date}"/>
</sql:query>
```

Tag attributes

The `<fmt:formatDate>` tag's attributes are as follows:

Attribute	Description	Required	Default
`value`	Date value to display	Yes	*None*
`type`	`DATE`, `TIME`, or `BOTH`	No	`date`
`dateStyle`	`FULL`, `LONG`, `MEDIUM`, `SHORT`, or `DEFAULT`	No	`default`
`timeStyle`	`FULL`, `LONG`, `MEDIUM`, `SHORT`, or `DEFAULT`	No	`default`
`pattern`	Custom formatting pattern	No	*None*
`timeZone`	Time zone of the displayed date	No	*Default time zone*
`var`	Name of the variable to store the formatted date (as a text string)	No	*Print to page*
`scope`	Scope of the variable to store the formatted date	No	`page`

The `<fmt:parseDate>` tag's attributes are as follows:

Attribute	Description	Required	Default
`value`	Date value to read (parse)	No	*Body*
`type`	`DATE`, `TIME`, or `BOTH`	No	`date`
`dateStyle`	`FULL`, `LONG`, `MEDIUM`, `SHORT`, or `DEFAULT`	No	`default`
`timeStyle`	`FULL`, `LONG`, `MEDIUM`, `SHORT`, or `DEFAULT`	No	`default`
`parseLocale`	Locale to use when parsing the date	No	*Default locale*
`pattern`	Custom parsing pattern	No	*None*

Attribute	Description	Required	Default
`timeZone`	Time zone of the parsed date	No	*Default time zone*
`var`	Name of the variable to store the parsed date (as a `java.util.Date`)	No	*Print to page*
`scope`	Scope of the variable to store the parsed date	No	`page`

The `<fmt:timeZone>` tag's attribute is as follows:

Attribute	Description	Required	Default
`value`	Time zone to apply to the body (`string` or `java.util.TimeZone`)	Yes	*None*

The `<fmt:setTimeZone>` tag's attributes are as follows:

Attribute	Description	Required	Default
`value`	Time zone to expose as a scoped or configuration variable	Yes	*None*
`var`	Name of the variable to store the new time zone	No	*Replace default*
`scope`	Scope of the variable to store the new time zone	No	`page`

A.5.3 Other internationalization

To assist with customized internationalization of applications, JSTL offers the following tags: `<fmt:setLocale>` to specify a new default locale, `<fmt:bundle>` and `<fmt:setBundle>` to prepare resource bundles for use, and `<fmt:message>` and `<fmt:param>` to output localized messages.

Examples

```
<fmt:setLocale value="en_US"/>
<fmt:setBundle basename="vulgarInsults"/>

<fmt:bundle basename="org.apache.bookies">
  <fmt:message key="threat" >
    <fmt:param value="${address}"/>
    <fmt:param value="${numberOfChildren}"/>
    <fmt:param value="${nameOfSpouse}"/>
  </fmt:message>
</fmt:bundle>
```

Tag attributes

The `<fmt:bundle>` tag's attributes are as follows:

Attribute	Description	Required	Default
`basename`	Base name of the resource bundle family to use in the body	Yes	*None*
`prefix`	Value to prepend to each key name in `<fmt:message>` subtags	No	*None*

The `<fmt:setBundle>` tag's attributes are as follows:

Attribute	Description	Required	Default
`basename`	Base name of the resource bundle family to expose as a scoped or configuration variable	Yes	*None*
`var`	Name of the variable to store the new bundle	No	*Replace default*
`scope`	Scope of the variable to store the new bundle	No	`page`

The `<fmt:message>` tag's attributes are as follows:

Attribute	Description	Required	Default
`key`	Message key to retrieve	No	*Body*
`bundle`	Resource bundle to use (JSTL `LocalizationContext`; see appendix B)	No	*Default bundle*
`var`	Name of the variable to store the localized message	No	*Print to page*
`scope`	Scope of the variable to store the localized message	No	`page`

The `<fmt:param>` tag's attribute is as follows:

Attribute	Description	Required	Default
`value`	Value of the parameter to set	No	*Body*