JavaServer Pages Standard Tag Library (JSTL)

Dean Wette

Senior Software Engineer, Educator
Object Computing, Inc.
St. Louis, MO

St. Louis Java Users Group, 14 November 2002

Architecture

- Model-View-Controller (MVC) Architecture
  - an industry proven pattern for separating major areas of functionality
    - Model - data access and domain objects that represent data
    - View - responsible for presentation to user
    - Controller - requests data and performs operations on them
  - Models, Views, and Controllers
    - loosely-coupled, with well-defined interfaces
  - globalize design, localize implementation
    - do not expose implementation details beyond interfaces & contracts
    - implementation changes should not propagate through system
- Problems with traditional Servlets and JSPs
  - HTML embedded in Java code (Servlets)
  - Java code embedded in JavaServer Pages (JSPs)
  - result is lowered tolerance to change

A Basic Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet {
   protected void doGet(HttpServletRequest req, HttpServletResponse res)
                            throws IOException, ServletException {
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();

      // "name" comes from the HTTP GET request
      // e.g. http://localhost:8080/servlets/hello?name=Dean
      String nameParam = req.getParameter("name");
      if (nameParam == null) nameParam = "World";

      out.println("<html>");
      out.println("<head><title>HelloWorld Output</title></head>");
      out.println("<body>");
      out.println("<h1>Hello, " + nameParam + "!</h1>");
      out.println("</body>");
      out.println("</html>");
   }
}
```

Servlets and XSLT

```
public class XSLTServlet extends HttpServlet {
   protected void doGet(HttpServletRequest req, HttpServletResponse res)
                        throws IOException, ServletException {
        res.setContentType("text/html");
        try {
            // get the data as XML DOM document
            Document coursesXml = ...
            Source xml = new DOMSource(coursesXml);

            TransformerFactory factory = TransformerFactory.newInstance();
            // get the XSLT stylesheet
            URL xsltURL = getServletContext().getResource("/courses.xsl");
            Source stylesheet = new StreamSource(xsltURL.toString());

            Transformer transformer = factory.newTransformer(stylesheet);
            // result of transform goes to HTTP response
            Result result = new StreamResult(res.getWriter());
            // emit result of XML/XSLT transformation
            transformer.transform(xml, result);
        } catch (Exception e) {
            throw new ServletException(e);
        }
    }
}
```

A JavaServer Page (JSP)

```
<%@ page import="java.util.*" %>
<jsp:useBean id="employee" class="Employee"/>
<jsp:setProperty name="employee" property="id"/>
<html>
  <head><title>
    <jsp:getProperty name="employee" property="fullName"/>'s Phone Numbers
  </title></head>
  <body>
    <table>
      <tr><th>Type</th><th>Number</th></tr>
      <!-- use a JSP scriptlet to iterate over a Map -->
      <% Iterator iter = employee.getPhones().entrySet().iterator();
         while (iter.hasNext()) {
            Map.Entry entry = (Map.Entry)iter.next(); %>
            <tr>
              <td><%= entry.getKey() %></td>
              <td><%= entry.getValue() %></td>
            </tr>
      <% } %>
    </table>
  </body>
<html>
```

A JSP With Custom Tags

```
<%@ page import="java.util.*" %>
<%@ taglib uri="/oci" prefix="oci" %>

<jsp:useBean id="employee" class="Employee"/>
<jsp:setProperty name="employee" property="id"/>
<html>
  <head><title>
    <jsp:getProperty name="employee" property="fullName"/>'s Phone
   Numbers
  </title></head>
  <body>
    <table>
      <tr><th>Type</th><th>Number</th></tr>
      <!-- use a JSP custom tag to iterate over a Map -->
      <oci:mapEntryIter map="<%= employee.getPhones() %>"/>
    </table>
  </body>
<html>
```

JSTL Overview

- Provides page authors with script-free environment
  - eliminates the need to use JSP scriptlets and expressions
  - uses a higher-level syntax for expressions

- Includes standard library of actions that implement common functionality
  - branching and looping control structures
  - URL rewriting, URL encoding, imports, redirects
  - XML parsing, flow control, transformations
  - database access using SQL
  - internationalization (I18n) and formatting

JSTL Overview (cont'd)

- JavaServer Pages Standard Tag Library (JSTL)
  - Java Community Process, Java Specification Request (JSR) 52
    - http://jcp.org/aboutJava/communityprocess/review/jsr052/
  - requires Servlet 2.3/JSP 1.2 compliant container
    - such as Tomcat 4
- Reference implementation (binaries and source) available from the Jakarta Project
  - http://jakarta.apache.org/taglibs/doc/standard-doc/intro.html

JSTL Tag Libraries

- Partitioned into multiple TLDs by functionality

Functionality	URI	Prefix
Core	http://java.sun.com/jstl/core	c
XML Processing	http://java.sun.com/jstl/xml	x
I18n capable formatting	http://java.sun.com/jstl/fmt	fmt
relational db access (SQL)	http://java.sun.com/jstl/sql	sql

Expression Language (EL)

- What is EL?
  - alternative to JSP scriptlets and rtexprvalues
  - simplifies data access/manipulation for page authors
    - hides details of page language
    - easy to use
    - custom tags can be written to support EL
  - syntax conforms to XML, inspired by ECMAScript and XPath
  - EL specification defined by JSR-152 (JSP 2.0) expert group
- EL available only in attribute values
  - JSP 2.0 will support EL in template text
  - attributes use literal expressions evaluated by tags at runtime
    - instead of rtexprvalues (unless using the _rt libraries)

      `<c:out value="${employee.SSN}"/>` instead of

      `<c:out value="<%= employee.getSSN()%>"/>`

EL Syntax

identifiers in EL expressions refer to:
PageContext.findAttribute(*identifier*)

- **${expr}**
  - used only in attribute values
  - may be mixed with static text
  - expressions may be combined to form larger expressions

- Examples

```
 <c:out value="Name: ${employee.name}"/>

<c:forEach var="employee" varStatus="status" items="${employees}">
  <c:out value="Employee ${status.count}: ${employee.name}"/>
</c:forEach>

<!-- addresses is a Map object -->
<c:out value="${user.addresses['billTo']}"/>
```

```
Name: Joe Smith
Employee 1: Joe Smith
Employee 2: Jane Doe
123 Main St.
New York, NY 10011
```

EL Operators

- Nested properties and collections
  - dot (`.`) operator used for accessing object properties
    - must follow JavaBean naming conventions

```
<c:out value="${employee.address.city}"/>
```

  - subscript (`[]`) operator for collection access
    - supports Collection, Map, and array types

see the spec for more details about collection support

aCollection or array:

```
<c:out value="${employees[0]}"/>
```

Map:

```
<c:out value="${salaries[employee.title]}"/>
```

`salaries` is a map, and the `title` property of `employee` is a map key

Operators (cont'd)

- Relational operators

  `==, !=, <, >, <=, >=`

  - aliases: `eq`, `ne`, `lt`, `gt`, `le`, `ge`
- Arithmetic operators:

  `+, -, *, /, %`

  - aliases: `div`, `mod`
- Logical operators

  `&&, ||, !`

  - aliases: `and`, `or`, `not`
- `empty` determines if value is `null` or empty

  ```
  <c:if test="${empty param['name']}">...</c:if>
  ```

enterCustomer.jsp

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
  <head>
    <title>Customer Data</title>
  </head>
  <body>
    <h1>Customer Data</h1>
    <c:if test="${param['showerror'] eq 'yes'}">
      <p><font color="red">
        <c:out value="Name is required!"/>
      </font></p>
    </c:if>
    Enter information about a customer:
    <form method="POST" action="submitCustomer.jsp">
```

enterCustomer.jsp (cont'd)

```
      <table>
        <tr>
          <td>Name: </td>
          <td><input type="text" name="name"/></td>
        </tr>
        <tr>
          <td>Phone: </td>
          <td><input type="text" name="phone"/></td>
        </tr>
        <tr>
          <td>Email: </td>
          <td><input type="text" name="email"/></td>
        </tr>
      </table>
      <input type="submit" value="Submit"/>
    </form>
  </body>
</html>
```

submitCustomer.jsp

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

  <c:if test="${empty param.name}">
    <c:redirect url="enterCustomer.jsp">
      <c:param name="showerror" value="yes"/>
    </c:redirect>
  </c:if>

  <sql:update dataSource="${applicationScope.dataSource}">
    INSERT INTO CUSTOMERS VALUES(?,?,?)
    <sql:param value="${param['name']}"/>
    <sql:param value="${param['phone']}"/>
    <sql:param value="${param['email']}"/>
  </sql:update>
```

submitCustomer.jsp (cont’d)

```
<html>
   <head>
     <title>Customer Data</title>
   </head>
   <body>
     <h1>Customer Data</h1>
     Here is the information you submitted:
     <table>
       <tr>
         <td>Name: <c:out value="${param['name']}"/></td>
       </tr>
       <tr>
         <td>Phone: <c:out value="${param['phone']}"/></td>
       </tr>
       <tr>
         <td>Email: <c:out value="${param['email']}"/></td>
       </tr>
     </table>
   </body>
 </html>
```

Core Tags Overview

- 4 categories of core tags
  - general purpose tags
    - for displaying values of EL expressions
    - for setting value of JSP scoped variables from EL expressions
  - conditional tags using if- and switch-like structures
  - iterator tags
    - for working with collections, arrays, and delimited strings
    - supports all `java.lang.util` collection types
    - supports `Iterator` and `Enumeration`
  - URL related actions

General Purpose Tags

- <c:out>
  - outputs result of expression evaluation as String to JspWriter object

```
<c:out value="${employee.name}"/>
<c:out value="${employee.homePhone}">
  none specified
</c:out>
```

“none specified” (default) is printed if `${employee.homePhone}` is null

- <c:set>
  - sets a JSP scoped variable to result of expression evaluation

```
<c:set value="${employee.name}" var="name"/>
<c:set var="name" scope="request">
  <c:out value="${employee.name}"/>
</c:set>
```

  - alternate syntax sets property of a target object to result of expression evaluation

```
<c:set value="Fred Bird" target="${employee}"
      property="name"/>
```

Conditional Tags

- <c:if>
  - simple conditional, executes body only if test is true

```
<c:if test="${employee.term ge 5}" var="vested"/>
<c:if test="${vested}">
  <li><c:out value="${employee.name}"/></li>
</c:out>
```

Conditional Tags (cont'd)

- <c:choose>
  - mutually exclusive conditional, executes body from a number of possible options

    ```
    <c:choose>
      <when> and <otherwise> subtags
    </c:choose>
    ```

    - processes body content of first subtag to evaluate true
- <c:when> - subtag of `<c:choose>`

  ```
  <c:when test="condition">body content</c:when>
  ```

- <c:otherwise> - optional last subtag of `<c:choose>`

  ```
  <c:otherwise>body content</c:otherwise>
  ```

<c:choose> Example

```
<c:set var="title" value="${employee.title}"/>
<c:choose>
  <c:when test="${title eq 'Junior Engineer'}">
    <c:set var="rate" value="10.00"/>
  </c:when>
  <c:when test="${title eq 'Senior Engineer'}">
    <c:set var="rate" value="20.00"/>
  </c:when>
  <c:when test="${title eq 'Principal Engineer'}">
    <c:set var="rate" value="30.00"/>
  </c:when>
  <c:otherwise>
    <c:set var="rate" value="25.00"/>
  </c:otherwise>
</c:choose>
```

Iterator Tags

- <c:forEach>
  - repeats nested body content over a collection of objects or for a specified number of times
  - syntax 1: iterate over collection or subset of collection

```
<c:forEach [var="varName"] items="collection"
           [varStatus="statusName"]
           [begin="begin"] [end="end"] [step="step"]>
  body content
<c:forEach>
```

  - syntax 2: iterate specified number of times

```
<c:forEach [var="varName"][varStatus="statusName"]
           begin="begin" end="end" [step="step"]>
  body content
<c:forEach>
```

<c:forEach> Example

```
<table>
  <tr><th>Number</th><th>Name</th><th>Title</th></tr>
  <c:forEach var="employee" items="${employees}" varStatus="status">
    <tr>
      <td><c:out value="${status.count}"/></td>
      <td><c:out value="${employee.name}"/></td>
      <td><c:out value="${employee.title}"/></td>
    </tr>
  </c:forEach>
</table>

<!-- say hello 3 times -->
<c:forEach begin="1" end="3">
  Hello, World!
</c:forEach>
```

LoopTagStatus

employee and status have "nested" visibility

URL Tags Overview

- Support for common JSP functionality
  - importing static and dynamic resources using `<c:import>`
    - like `<jsp:include>` but
      - allows importing from foreign contexts
      - supports protocols other than HTTP (e.g. ftp)
      - more efficient, avoids unnecessary buffering
    - can also import resource into String or Reader object
      - String is cached and supports reuse
  - URL rewriting for session tracking using `<c:url>`
  - HTTP redirects using `<c:redirect>`
  - hypertext links with URL encoding using `<c:param>`
    - as a nested action for `<c:url>`, `<c:import>`, and `<c:redirect>`
  - specification has more detail about behavior of these actions

<c:import>

- Imports resources into page
  - 3 types of URLs can be specified
    - absolute, with beginning protocol (e.g. http://...)
    - relative within same context
    - relative within a foreign context
  - optional `<param>` subtags are used to encode query string parameters

  - import resource with absolute URL

```
<c:import url="http://www.oci.com/menu.html"/>
```

  - import resource with relative URL, export to a variable

```
<c:import url="/employees" var="engineers">
    <c:param name="title" value="Software Engineer"/>
<c:import>
```

Other URL Tags

- <c:url> - performs URL rewriting
  - optional subtags are used to encode query parameters

```
<c:url value="/employees" var="link">
  <c:param name="mode" value="list"/>
</c:url>
<a href='<c:out value="${link}"/>'>List Employees</a>
```

- <c:redirect> - sends a HTTP redirect to the client
    - aborts further processing of page (i.e. `Tag.SKIP_PAGE`)
  - follows same url rewriting rules as `<c:url>`
  - optional subtags are used to encode query parameters

```
<c:redirect url="/employees">
  <c:param name="mode" value="list"/>
</c:redirect>
<c:redirect url="http://java.sun.com/index.html"/>
```

XML Tags Overview

- Allows page authors to easily access XML content
  - supports XPath expressions for document traversal
  - includes XPath function library
- Based on XPath expression language
  - variable bindings for XPath expression provide access to JSP scoped attributes
  - mappings between Java and XPath types
- Three categories of XML tags
  - core XML support
    - XML parsing
    - output XPath expressions
    - export XPath expressions to JSP scoped attributes
  - flow control corresponding to core conditional and iteration tags
  - XSL transformations

Core XML Tags

- <x:parse>
  - parses an XML document
- <x:out>
  - evaluates an XPath expression and prints the result
- <x:set>
  - evaluates an XPath expression and stores the result
- Examples to follow…

XML Flow Control Tags

- Provides XML/XPath counterparts to core tags
  - uses XPath expressions for accessing XML
- Conditional processing tags
  - if, choose, when, otherwise
  - syntax is similar to core actions

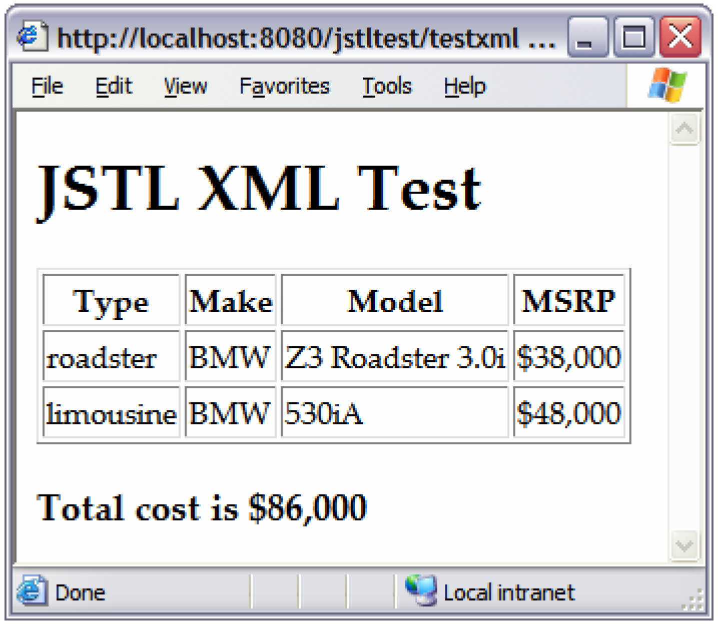
    ```
    <x:if select="XPathExpr" [var="varname" [scope=...]] />
    ```

    - also `<x:choose>`, `<x:when>`, `<x:otherwise>`
- Iterative processing tag

```
<x:forEach select="XPathExpr" [var="varName"]>
  body content
</x:forEach>
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<cars>
  <car type="roadster">
    <make>BMW</make>
    <model>Z3 Roadster 3.0i</model>
    <price>38000</price>
  </car>
  <car type="limousine">
    <make>BMW</make>
    <model>530iA</model>
    <price>48000</price>
  </car>
</cars>
```


http://localhost:8080/jstltest/testxml ...
File Edit View Favorites Tools Help
Done
Local intranet

JSTL XML Test

Type	Make	Model	MSRP
roadster	BMW	Z3 Roadster 3.0i	$38,000
limousine	BMW	530iA	$48,000

Total cost is $86,000

XML Example (cont'd)

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

  <c:import url="/car.xml" varReader="reader">
    <x:parse xml="${reader}" var="cars"/>
  </c:import>

  <c:set var="pattern" value="$#,##0"/>

  <html>
    <body>
      <h1>JSTL XML Test</h1>
      <table border="1">
        <tr>
          <th>Type</th>
          <th>Make</th>
          <th>Model</th>
          <th>MSRP</th>
        </tr>
```

import XML doc into a Reader object and parse it

XML Example (cont'd)

```
        <x:forEach select="$cars//car">
          <x:set var="price" select="number(price)"/>
          <tr>
            <td><x:out select="@type"/></td>
            <td><x:out select="make"/></td>
            <td><x:out select="model"/></td>
            <td><fmt:formatNumber value="${price}"
                                      pattern="${pattern}"/></td>
          </tr>
        </x:forEach>
      </table>
      <x:set var="total" select="sum($cars//price)"/>
      <h3>Total cost is
          <fmt:formatNumber value="${total}" pattern="${pattern}"/>
      </h3>
    </body>
  </html>
```

XML Transform Actions

- Provides support for XSLT transforms in JSP pages
  - can be used to mix JSP and XSLT for page generation

- <x:transform>
  - transforms an XML document using XSLT
- <x:param>
  - sets transformation (stylesheet) parameters

Transform Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:param name="pattern" select="'$#,##0'"/>
  <xsl:template match="/">
    <table border="1">
      <tr>
        <th>Type</th>
        <th>Make</th>
        <th>Model</th>
        <th>MSRP</th>
      </tr>
      <xsl:for-each select="//car">
        <tr>
          <td><xsl:value-of select="@type"/></td>
          <td><xsl:value-of select="make"/></td>
          <td><xsl:value-of select="model"/></td>
          <td><xsl:value-of select="format-number(price, $pattern)"/></td>
        </tr>
      </xsl:for-each>
    </table>
  </xsl:template>
</xsl:stylesheet>
```

Transform Example (cont'd)

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jstl/xml" prefix="x" %>
  <c:set var="pattern" value="$#,##0"/>
  <c:import url="/car.xml" varReader="reader">
    <x:parse xml="${reader}" var="carxml"/>
  </c:import>
  <c:import url="/car.xslt" var="carxslt"/>
  <x:set var="total" select="sum($carxml//price)"/>
  <html>
    <body>
      <h1>JSTL XSLT Test</h1>
      <x:transform xslt="${carxslt}" xml="${carxml}">
        <x:param name="pattern" value="${pattern}"/>
      </x:transform>

      <h3>Total cost is <fmt:formatNumber value="${total}"
                                          pattern="${pattern}"/></h3>
    </body>
  </html>
```

SQL Actions

- For interacting with relational databases
- Best for quick prototyping
  - scalable web apps generally implement separate data access layer
- Database drivers can be specified directly, or using a DataSource via JNDI
- Supports four types of operations
  - execute database queries (select) with `<sql:query>`
  - access query results with `Result` object and core iterator tags
  - execute database updates (insert, update, delete) with `<sql:update>`
- Queries can be part of a transaction
- SQL statements specified in tag body

SQL Example

```
<sql:query var="engineers" dataSource="${dataSource}">

  SELECT * FROM employees
  WHERE title LIKE '%Engineer'
  ORDER BY lastname
</sql:query>

<table>
  <c:forEach var="row" items="${engineers.rows}">
    <tr>
      <td><c:out value="${row.lastname}"/></td>
      <td><c:out value="${row.firstname}"/></td>
      <td><c:out value="${row.address}"/></td>
    </tr>
  </c:forEach>
</table>
```

`javax.servlet.jsp.jstl.sql.Result`

maps column names to values

I18n and Formatting Actions

- Supports internationalizing web applications
  - messages, numbers, dates, time, currencies, etc.
- Uses resource bundles to localize text messages
  - locale specific properties located automatically using runtime locale information
- Utilizes Java's properties file and resource bundle mechanisms for locale lookup
  - see `java.util.ResourceBundle`
- Formatting and parsing actions are also useful for non-localized apps
  - from previous XML example

```
<c:set var="pattern" value="$#,##0"/>
...
<h3>Total cost is <fmt:formatNumber value="${total}"
                                    pattern="${pattern}"/></h3>
```

Formatting Examples

- Date and Time

```
<jsp:useBean id="now" class="java.util.Date"/>
<fmt:formatDate value="${now}" type="both"
                timeStyle="long" dateStyle="long"/>
```

  - Wednesday, May 1, 2002 11:18:28 AM CDT
  - Mittwoch, 1. Mai 2002 11.59 Uhr CDT

```
<fmt:formatDate value="${now}" pattern="MM/dd/yyyy"/>
```

  - 05/01/2002

- Numbers and Currency

```
<fmt:formatNumber value="10562.1" type="currency"/>
```

  - $10,562.10

```
<fmt:formatNumber value= "12849.36549" pattern="#,#00.0#"/>
```

  - 12,849.37