



JavaServer Pages: Tag Libraries vs. JavaBeans

A Whitepaper from Sybase, Inc.

*By Jason R. Weiss
Sr. Software Engineer
eBusiness Systems Group
Sybase, Inc.*

Table Of Contents

1. Overview.....	2
2. Custom Tag Library or JavaBean?	2
3. Code Example	2
4. JavaBeans.....	3
5. Custom Tag Libraries	5
6. How do I access the <code>request</code> object?.....	6
7. Comparison Chart.....	7

1. Overview

Java provides developers with JavaServer Pages (JSPs) and Servlets as a superior alternative to traditional CGI programs. The architecture of JSPs provide support for a logical and physical separation between the HTML page designers and the component developers, who specialize in implementing advanced business logic. Another advantage of a JSP is its ability to implement a custom tag library. These custom tags allow page designers to abstract themselves from a complex set of logic; include this tag and it will have this effect on the web page.

Although examples can be located detailing how to build a custom tag library, developers are rarely educated on how to make a proper design decision on which to use, a custom tag library or a JavaBean.

The intent of this paper is to discuss the rationale that developers should use when choosing between implementing a custom tag library or utilizing a JavaBean. This paper does not provide detailed and robust examples on how to build JavaBeans and tag libraries for use inside of a JSP, although a .zip file should have accompanied this document with a small, simple example of using both JavaBeans and custom tags inside of a JSP page. Additional white papers will cover implementation issues in a greater depth.

It should be noted that this paper assumes that the reader has some knowledge of using JSPs to implement a J2EE Web Application.

2. Custom Tag Library or JavaBean?

There is no standard HTML or JSP tag that is capable of executing different logic based on different inputs fed into the page. In order to create a *reactive* page, Java code becomes a necessity. Both custom tag libraries and JavaBeans can be utilized to separate presentation from complex business logic, providing the ability to run different logic branches. It is at this point that it becomes imperative for a developer to understand the idiosyncrasies, strengths, and weaknesses of each of these options in order to make educated design decisions. We will take a brief look at both options, and provide a set of heuristics in table-form that outline the capabilities and features of each. Use this table as a reference to help you decide which approach to use in your design, JavaBeans or a custom tag library.

3. Code Example

This white paper should have been distributed as a single zip file containing this Adobe Acrobat file, and another zip file that contains samples of a JavaBean, a Custom Tag Library, a JavaServer Page, and a J2EE WAR target. When the code example zip file is unzipped, be sure to extract all the files into an empty directory, and be sure to maintain the folder structure recorded inside of the zip. After extracted, use PowerJ to open up the `example` workspace.

4. JavaBeans

JavaBeans technology has been around for awhile. Full coverage of JavaBeans is well beyond our scope here, but if you are interested in learning more about JavaBeans, there are numerous books on the subject, as well as documentation from Sun at <http://java.sun.com/beans/docs/>.

JavaBeans are a useful option for encapsulating logic, and removing Java code from the middle of a JSP page. JavaBeans have been available for use inside of a JSP since the 1.0 JSP specification. JSPs are equipped with three dedicated tags that specialize in working with JavaBeans:

```
<jsp:useBean>
<jsp:getProperty>
<jsp:setProperty>
```

There are a number of attributes for each of these tags, which again are beyond our scope and will not be covered here. Suffice to say, each of these are empty tags (an empty tag has no text appearing between the opening and closing tags, and can be simplified to a single tag with a “/” terminal), and can not manipulate the contents of the JSP page in any way. To get a value out of the bean, the page designer declares their intention to use a bean first, for example:

```
<jsp:useBean id="myBean" class="com.some.company.MyBean" />
```

The id attribute is the name that the page designer will use to reference the bean. The class attribute is the class that should be associated with the id provided. A JavaBean may store a color code, and expose the value of that color code through a set of methods, for example:

- `String getColorCode()`
- `void setColorCode(String color)`

The bean is said to have exposed a property named `colorCode`. Read-only properties can be established by not defining a mutator method.

Once a bean is created through the `<jsp:useBean>` tag, page designers can access the bean properties elsewhere on the page. To work with the properties of a bean, the page designer uses either the `<jsp:getProperty/>` or `<jsp:setProperty/>` tag. For example, this tag would output the value of the color code to the JSP page:

```
<jsp:getProperty name="myBean" property="colorCode" />
```

JavaBeans force developers to respect the notion of encapsulation. The methods used to manipulate the properties on the beans could do anything behind the scenes, including accessing a database or implementing complex validation rules. In fact, reading and writing properties from a JavaBean could fire off different actions. It is because of encapsulation these JavaBeans can be thought of as “black-box” programming. Page designers can use these black boxes throughout the site without an iota of understanding of the complexities behind the values that mystically appear.

Here is a complete example of using a JavaBean inside of a JSP page:

```

<%@ page import="com.some.company.*"
      language="java"
      buffer="8kb"
      autoFlush="true"
      isThreadSafe="true"
      info="Copyright (C) 2001, Sybase, Inc."
      isErrorPage="false" %>

<jsp:useBean id="myBean" class="com.some.company.MyBean" />
<HTML>
<HEAD>
<TITLE>Some Page</TITLE>
</HEAD>
<BODY>
      Here is something from the bean:
      <jsp:getProperty name="myBean" property="colorCode" />
</BODY>
</HTML>

```

To reiterate a major point, the logic behind the method used to represent the `colorCode` property could be quite complex. For example, it might connect to a remote database or invoke a method running on an application server half-way around the world in order to find out the value of the property. Regardless of the approach, the page designer only needs to know the name of the property they are interested in using, not the details on how to obtain the value.

Although it is possible to invoke bean methods directly after declaring the bean using a `<jsp:useBean>` tag, the JSP 1.1 specification explicitly refers to the use of `<jsp:getProperty>` and `<jsp:setProperty>`. In other words, the specification is unclear about whether invoking methods directly is an acceptable practice or not. The safest course of action is to stick with these two tags, and avoid calling methods on the bean directly.

JavaBeans are great for handling business logic. The aforementioned JavaBean tags have the built-in ability to work hand in hand with data obtained through a web form. When setting a property on a JavaBean, page designers only need to remember to set the `property` attribute on the tag to the same name used inside of the form. The JSP will automatically assume that the value it should provide needs to come from the form that it is processing. In situations where the form name must be different from the property name on the bean, a combination of the `property` and `param` attributes can be set, thereby establishing a link between the two.

Bean tags also understand the notion of scope. Recall that scope provides an indication of where the bean is recognized. By specifying the optional `scope` attribute on the `<jsp:useBean/>` tag, the bean will only be instantiated if a matching `id` is not already present at the indicated scope level; page, application, session, or request. Although we won't go into the details between the different scopes here, this feature is unique to a JavaBean.

Finally, it is worth mentioning that our discussion on JavaBeans does not relate to Enterprise Java Beans (EJBs), which are part of J2EE. There are proposals to introduce another standard JSP tag that provides similar capabilities, allowing a JSP to establish a reference to an EJB on a remote server, but while this article is written there is no standard yet. Also noteworthy is the fact that each of these tags is case sensitive, and attributes must be enclosed in either single or double quotes.

5. Custom Tag Libraries

The JavaServer Pages 1.1 specification provides a mechanism for defining new actions into a JSP page. A **custom tag library** is comprised of one or more Java classes (called tag handlers), and an XML tag library description file (tag library, for short). The tag library dictates the new tag names and valid attributes for those tags.

Tag handler classes, together with a tag library, determine how the tags, their attributes, and their bodies will be interpreted and processed at request time from inside a JSP page. Collectively, they provide an architecture that is arguably more apt than a JavaBean at encapsulating a complex operation from the page designer. Unfortunately, that power comes with a cost; it takes more effort to build and implement a custom tag library than a simple JavaBean.

Here is an example of a JSP page using a small custom tag library:

```
<%@ page import="com.some.company.*"
      language="java"
      buffer="8kb"
      autoFlush="true"
      isThreadSafe="true"
      info="Copyright (C) 2001, Sybase, Inc."
      isErrorPage="false" %>

<jsp:useBean id="myBean" class="com.some.company.MyBean" />

<HTML>
<BODY>

Here is something from the bean:
<jsp:getProperty name="myBean" property="colorCode" />

<%@ taglib uri="WEB-INF/tlds/tablib.tld" prefix="example"%>
Here is the output from a custom tag:

<example:myCustomTag someColor="#FFFF00" />

</BODY>
</HTML>
```

At page request time, the Java classes associated with this custom tag are invoked, and the generated content is streamed back to the client in place of this tag. The classes that implement that tag library are distributed with the Web Application either in the web-inf/lib directory, contained in a single .JAR, or in the web-inf/classes directory, as individual classes. The taglib page directive provides directions on where the tag library exists. This tag library defines the implementation details of the tag, in XML format. The following tag library was referenced inside of the JSP page by the taglib page directive (<%@ taglib ...>), and it established the prefix of example for all the tags inside of the library.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
      PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
      "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">

<!-- This is my tab library descriptor -->
<taglib>
  <tlibversion>1.0</tlibversion>
```

```

<jspversion>1.1</jspversion>
<info>
  Custom Tag Example
</info>

<!-- Tags -->
<tag>
  <name>myCustomTag</name>
  <tagclass>com.some.company.ExampleTag</tagclass>
  <bodycontent>EMPTY</bodycontent>
  <info>Just a small sample tag</info>
  <attribute>
    <name>someColor</name>
    <required>false</required>
  </attribute>
</tag>
</taglib>

```

You can see the library exposes a tag called `myCustomTag` with a single, optional attribute, `someColor`. The tag library also indicates that the body content (`<bodycontent>`) is empty and should be ignored.

Tag libraries typically (when built correctly) provide the page designer with a more meaningful, self-descriptive definition of what the tag is doing, possibly demonstrating a self-documenting characteristic. Instead of being restricted to a tag that begins `<jsg:getProperty/>`, the page can use a tag like `<employee:PreferredColor/>`, making the HTML more legible down the road for maintenance purposes.

Similar to a JavaBean, page designers can inform a custom tag of a value by setting attributes on the tag. However, tag libraries also have the unique ability of allowing the tag developer to directly expose variables to the JSP page. For example, a tag library could populate a variable called `employee`, permitting the page designer to reference the variable elsewhere on the page inside of a scriptlet or expression, like this:

```
<%= employee.getPreferredColor() %>
```

6. How do I access the `request` object?

Both JavaBeans and tag libraries can access the various methods (and thereby data associated with) the `request` object. Tag library attributes can declaratively indicate that an attribute can be specified as an expression:

```
<rtexpression>true</rtexpression>
```

This means that the following syntax is valid for a custom tag:

```
<example:myTag
  myAttribute="<%= request.getContentLength() %>">
```

Similarly, if a JavaBean needs access to the same type of content, use the `<jsp:setProperty>` tag:

```
<jsp:setProperty
  name="myBean"
  property="someProperty"
  value="<%= request.getContentLenght() %>" />
```

7. Comparison Chart

The following chart provides a one-stop reference to help developers choose which approach is right for their problem, JavaBeans or Custom Tag Libraries.

	JavaBean	Custom Tag Library
Deployment Location	Accessible through the CLASSPATH on the server hosting the Web Application. Typically deployed as a .JAR file into the web-inf/lib directory.	Either as stand-alone classes deployed into web-inf/classes, or as a .JAR file inside of web-inf/lib.
Mappings	No concept of mapping the physical JavaBean JAR into a .WAR deployment descriptor.	The .WAR deployment descriptor can provide a logical mapping between the tag library descriptor URI and the physical tag library descriptor location. For example, the JSPs can reference /TagLibrary and the deployment descriptor will associate this with /web-inf/tlds/taglib.tld.
Tag Flow	Rigid; must use 1 of 3 well-defined tags to access the bean, each of which use the <code>jsp</code> prefix	Fluid; tag library author determines the names of the tags, the page designer determines the prefix
Variable Access	Page designers can only get/set values through the use of the standard bean tags: <pre><jsp:getProeprty/> <jsp:setProperty/></pre>	Page designers assign values through tag attributes, and tag developers can optionally expose custom variables for use elsewhere on the JSP page.
Scope	Can be defined at the page, application, session, or request level.	Page scope only; tag can only be used on a JSP page that references the tag library through the <code>taglib</code> page directive.
Manipulation of JSP Content	No	Yes
Ability to encapsulate complex business logic from the page designer	Yes	Yes
Ability to access the request object	Yes	Yes. Must remember to add the <code><rtexpression></code> tag for each attribute that needs this capability.
Ability to provide a logical name for a complex business operation	No	Yes, the name of the tag is developer defined, and the name of the prefix is page designer defined.
Ease of development	Relatively Simple	More Complex
Version Introduced	JSP 1.0	JSP 1.1 (EAServer 3.6.1 implements JSP 1.1)



Sybase, Inc. Worldwide Headquarters 6475 Christie Avenue, Emeryville, CA 94608 USA
Phone: 1-800-8-SYBASE (in US and Canada); Fax: 1-510-922-3210.

World Wide Web: <http://www.sybase.com>

Copyright © 1999 Sybase, Inc. All rights reserved. Sybase, the Sybase logo, Enterprise Application Server, PowerBuilder, and PowerJ are trademarks of Sybase, Inc. Java is a trademark of Sun Microsystems, Inc. Visual Edge, ObjectBridge, and Golden Gate are trademarks or registered trademarks of Visual Edge Software Corp. SAP™, R/3™, and all SAP™ products and service names referenced herein are trademarks or registered trademarks of SAP AG. All other trademarks are property of their respective owners. ® indicates registration in the United States. Specifications are subject to change without notice.