SIMPLY RICH

# ZK JSP Tags™ User Guide

**Version 1.2**

April 2008

**Potix Corporation**

# Table of Contents

# 1. Before You Start

## New to JSP

Because this user guide is written for developers who already familiar with JSP technology. So if you never use JSP before, please refer to these links below:

Sun's official J2EE1.4 tutorial(See CH12)

JSP Tutorial

## New to ZK

Because ZK JSP Tags is based on ZK and most usages and features in ZK JSP Tags are the same as those in ZK, so knowing how to use ZK will greatly help you in application development using ZK JSP Tags. To know how to use ZK, please refer to these links below:

ZK Quick Start Guide

ZK official Developer's Guide(online)

## What to Download

| File | Description |
|---|---|
| `ZK-3.0.4.zip` | The binary distribution of ZK. ZK JSP Tags is a JSP tag library which is based on ZK environment, so you will need to install ZK first (including all required Jar files and settings). |
| `Zk-JspTags-1.2.zip` | The binary distribution of ZK JSP Tags. |

# 2. Installation

### Install ZK to Your Web Archive First.

Please follow ZK QuickStart Guide to install the ZK environment first. The document link:

### Install ZK JSP Tags

Installation of ZK JSP Tags is very simple; you unpack the downloaded zip file, copy zuljsp.jar to WEB-INF/lib/, do some little configuration in web.xml and you can declaring ZK component tag in your JSP document. We'll discuss how to do settings in web.xml below.

### Additional Configurations for ZK JSP Tags in web.xml

You have to do one more configuration in web.xml to make ZK JSP Tags working for you. You must add a Listener(`org.zkoss.jsp.spec.JspFactoryContextListener`) into web.xml as following:

```
<listener>
    <description>ZK JSP Tags environment initiation </description>
    <display-name>ZK JSP Initiator</display-name>
    <listener-class>org.zkoss.jsp.spec.JspFactoryContextListener</listener-class>
</listener>
```

# 3. Declare ZK JSP tags in your JSP Document

## Taglib Declaration

Using ZK JSP Tags is the same as to using other JSP tag libraries. You declare a <%@taglib %> to define the prefix and mapping URI of the ZK JSP Tags:

```
<%page uri="http://www.zkoss.org/jsp/zul" prefix="zk" %>
```

## Page Tag Declaration

In ZK world, every ZK components must be contained by a Desktop instance. It will manage the Ajax connections of each component. In JSP, you have to specify a <page> tag to initialize the Desktop instance if it's not existed:

```
<zk:page zscriptlanguage="java" >
...
// The ZK Component Tags declaration Area...
...
</zk:page>
// Wrong place to put ZK Component Tags...
```

The page tag also create a Page instance automatically, and this page instance is a container to contain components. In one JSP document, you can declare multiple pages like this:

```
<zk:page  id="pg1" style="height:100px;width=300px">
// The ZK Components declaration Area...
</zk:page>

//Other JSP Content...

<zk:page  id="pg2" style="height:100px;width=300px">
// The ZK Components declaration Area...
</zk:page>
```

The rule to access variables across pages is the same as to do so in pure ZK document. For more detailed information about Page(`org.zkoss.zk.ui.Page`), please refer to ZK Developer's Guide.

## Component Tag Declaration

After declaring page tag, now we can add some component tags into it. For example:

```
<zk:page>
    <zk:window id="myWin" title="ZK JSP Tags Demo">
        <zk:label value="Hello ZK JSP Tags!!!"/>
    </zk:window>

    <h1>mix with other tags.</h1>
```

```
</zk:page>
```

As you can see, the codes looks like the ZUL document declaration.

## Transform Your ZK Component Set to ZK JSP Tags

Currently, ZK JSP Tags support all ZUL components. If you want to make your own ZK component set also JSP tag library, the process is quite simple:

1. New a Java class as following:

```
public class ButtonTag extends org.zkoss.jsp.zul.impl.BranchTag
{
      protected String getJspTagName(){// That's quite simple!!!
          return "button";
      }
}
```

   It's really simple. ZK will invoke getJspTagName() method to get the tag name, and use that name to search through ZK component definition to find out what the real ZK component Java class is and how to initialize and new an associated instance object.

   **Note:** You must guarantee the tag name could be found in your ZK environment.

2. Follow the JSP Tag definition process, declare this JSP tag in tld, for example:

```
<tag>
<name>button</name>
<tag-class>org.zkoss.jsp.zul.ButtonTag</tag-class>
<body-content>scriptless</body-content>
<dynamic-attribute>true</dynamic-attribute>
<attribute>
    <name>if</name>
    <required>false</required>
    <rtexprvalue>true</rtexprvalue>
</attribute>
...
</tag>
```

   Although dynamic attribute can be set true and all kinds of attributes could be declared in tag, you still need to add these static attributes in your JSP tag definition. They are: if, unless, use, forward.

3. Use tld in JSP document with <%@ taglib%>

   Follow normal JSP tag library declaration, use your custom ZK JSP Tags in JSP document.

# 4. Expression Language(EL) in ZK JSP

## Introduction

In JSP, EL is a very useful technology to bind bean variable, evaluating simple expression and invoke predefined static Java functions. ZK JSP Tags EL environment provides you fully accessibility to access variables both in JSP world and ZK world in both JavaEE 4 and JavaEE 5 standards. If you followed the steps from the chapter 2 "Installation" , your ZK JSP Tags EL environment will start up automatically.

## Use ZK Components as a Bean Variable in EL

Please take a look at the sample code below:

```
<zk:page>
    <zk:window id="myWin" title="My window's title.">
    </zk:window>

    <h1>the window ${myWin.id}'s title is:${myWin.title}</h1>
</zk:page>
```

In JSP EL, you can access ZK components as Java beans using its id. And even you want to use your own Java class is still okay:

```
<zk:page>
    <zk:window id="myWin" use="org.zkoss.jspdemo.MyWindow" myValue="this is value">
        <h1> ${self.myValue}"<h1/>
    </zk:window>

    <h1>the window ${myWin.id}'s title is:${myWin.title}</h1>
</zk:page>
```

As you can see the <h1> tag inside myWin, you can use ZK keyword "self" to represent current ZK component.

## Use zscript Variable in EL

The most powerful feature in ZK JSP EL is that you can also access zscript variables directly:

```
<zk:page>
    <zk:zscript>
         String zsStr = "This is a zscript variable";
    </zk:zscript>

    <zk:label value="${zsStr}"/>
</zk:page>
```

## ZK ID Space Concept in EL

In pure ZK, the variable accessibility scope is based on the concept of ID Space(If you don't know what ID Space is for, please refer to ZK Developer's Guide). In ZK JSP environment, it is totally the same as the pure ZK. For example:

```
<zk:page>
    <zk:zscript>
         String A = "This is page scope A";
    </zk:zscript>
    <zk:window id="myWin">
        <zk:zscript>
             String A = "This is myWin's A";
        </zk:zscript>
        <zk:label value="${A}" id="innerLabel"/>
    </zk:window>
    <zk:label value="${A}" id="outerLabel"/>
</zk:page>
```

According to the result, the **innerLabel** will show "This is myWin's A", and the **outerLabel** will show "This is page scope A".

## Access JSP Environment Variable in zscript

We have shown you how to access ZK variables in EL, but how about accessing JSP variables in ZK? For example, how to access a JSP scriptlet variable? How to access JSTL's foreach iterator variable?

According to JSP specification, if you want to use scriptlet variable in EL or any non scriptlet area, you must put them into request, session or application's attributes. Like this:

```
<%
    String myStr = "scriptlet String";
    request.setAttribute("myStr",myStr);
%>
<zk:page>
    <zk:label value="${myStr}"/>// OK!
    <zk:zscript>
        str1 = "use " + requestScope.get("myStr")+ " in zscript.";// OK!
        str2 = "use " + myStr + " in zscript.";// will cause failure!
        // Because ZK doesn't allow direct access to the request attributes.
    </zk:zscript>
</zk:page>
```

As you can see, you can get those variables back to zscript area using **requestScope**, **sessionScope** and **applicationScope**. To know more about how to use these implicit ZK keywords, please refer to ZK Developer's Guide.

Please be careful about using those implicit objects and their life cycles. For example, following example will show "null" instead of "This is scriptlet string.":

```
<%
    String myStr = "This is scriptlet string.";
    request.setAttribute("myStr",myStr);
%>
<zk:page>
    <zk:label value="${myStr}"/>// OK!
    <zk:button label="push me!">
        <zk:attribute name="onClick">
            alert("str is:"+requestScope.get("myStr"));// will show null!
        </zk:attribute>
    </zk:button>
</zk:page>
```

Notice that when the button onClick event is triggered (which is an independent request), the previous request which stores “myStr” was already gone, and the requestScope is the onClick request one;  not the old one.

Sometimes in JSTL foreach tag, we want to store the variable of iterator in generated ZK components, here the <custom-attribute ...> is an approach:

```
<%
    String[] strArr = new String[5];
    //init myStrs...
    request.setAttribute("strArr", strArr);
%>
<zk:page>
    <c:forEach var="myStr" items="${strArr}">
        <zk:button label="push me!">
            <custom-attribute myStr="${myStr}"/>// use custom attribute to store...
            <zk:attribute name="onClick">
                alert("str is:"+self.getAttribute("myStr"));// show result
            </zk:attribute>
        </zk:button>
    </c:foreach>
</zk:page>
```

# 5. Processing Instruction Tags

In pure ZK document we can declare many kinds of processing instructions to define components, initiators and variable resolvers. These directives are also supported in ZK JSP Tags. The place to declare directives is outside the <zk:page> tags which should be put at the topmost of whole document. For example:

```
<zk:init .../>
<zk:component .../>
<zk:variable-resolver .../>
...
<zk:page>
...
</zk:page>
```

## The init Directive

```
<zk:init use="" [arg0=""] [arg0=""] [arg0=""] .../>
```

The init directive is the same as the ZK init directive. The deferent part is in JSP we don't use class attribute, we use "use" attribute. The Java class represented by use attribute must implement `org.zkoss.zk.ui.util.Initator` interface. The "zscript" attribute is not implemented yet.

For more detailed information, please refer to : ZK Developer's Guide.

## The variable-resolver Directive

```
<zk:variable-resolver use="..." />
```

Specifies the variable resolver that will be used by the `zscript` interpreter to resolve unknown variables. The specified class must implement the `org.zkoss.xel.VariableResolver` interface.

For more detailed information, please refer to : ZK Developer's Guide.

## The component Directive

```
<zk:component name="myName" macroURI="/mypath/my.zul"
  [prop1="value1"] [prop2="value2"].../>

<Zk:component name="myName" [class="myPackage.myClass"]
  [extends="existentName"] [moldName="myMoldName"] [moldURI="/myMoldURI"]
  [prop1="value1"] [prop2="value2"].../>
```

Defines a new component for a particular page. Components defined in this directive are visible to the pages which are declared in this JSP document. To define components that can be used in any page, please refer to ZK language addon, which is a XML file defining components for all pages in a Web Application.

The detailed information about component directive, please refer to the **6. Component Page Definition** below.

## The Unsupported Directives in ZK JSP

Currently unsupported directives are: `import`, `link`, `xel-method`, `page`. Some of them is not supported because JSP has already provided such features.

# 6. Component Page Definition

The ZK Component page definition provides the most powerful customizable ability to let us define a new kind of component tag dynamically. However, because the limitation of the JSP specification, we cannot declare a new JSP tag name dynamically. To overcome such limitations in ZK JSP Tags we invent a special <zk:ui> tag to by-pass it. For more detailed information please look the section of **The Usage of `ui` Tag** below.

## By-class Format Component Definition

```
<zk:component name="componentName" [class="myPackage.myClass"]
  [extends="existentJavaClassName"] [moldName="myMoldName"] [moldURI="/myMoldURI"]
  [prop1="value1"] [prop2="value2"].../>
```

Creates a new component definition which will be used in the JSP document. If the class attribute is being declared, it must implements the `org.zkoss.zk.ui.Component` interface.

## Macro Component Definition

```
<zk:component name="componentName" macroURI="/mypath/my.zul"
   [prop1="value1"] [prop2="value2"].../>
```

Defines a new component based on a ZUL document. It is called *a macro component*. In other words, once an instance of the new component is created, it creates child components based on the specified ZUL document (the `macroURI` attribute). For more details, refer to the **ZK developer's Guide: Macro Components** chapter.

## Inline Macro Definition

```
<zk:component name="componentName" macroURI="/mypath/my.zul" [inline="true"]
  [prop1="value1"] [prop2="value2"].../>
```

Inline macro is a way to let you attach a component set defined in the macro URI which denoted a ZUL document. In other words, it works the same as if you copy the content from the macro URI directly to the target document.

## The Usage of `ui` Tag

In order to use the dynamically declared component in JSP, ZK use a spacial <zk:ui> tag to accomplish this approach. For example:

```
<zk:component name="username" inline="true" macroURI="/macro/username.zul">
...
<zk:page>
  <zk:ui tag="username" who="ZK User"/>

</zk:page>
```

# Appendix A. ZK JSP Functionalities List

## Supporting Status

This appendix shows the status of ZK JSP Tags implementation of ZK specification. The meaning of the status symbol is as bellow:

**O** means fully support.

**X** means not support.

***** means partial support or supported in deferent way.

## Functionalities List

| Topics | Items | ZK JSP | Note |
|---|---|---|---|
| Implicit Objects | applicationScope - java.util.Map | O | |
| | arg - java.util.Map | O | |
| | componentScope - java.util.Map | O | |
| | desktop - org.zkoss.zk.ui.Desktop | O | |
| | desktopScope - java.util.Map | O | |
| | each - java.lang.Object | * | Use JSTL **forEach** instead |
| | event - org.zkoss.zk.ui.event.Event or derived | O | |
| | forEachStatus – org.zkoss.zk.ui.util.ForEachStatus | * | Use JSTL **forEach** instead |
| | page - org.zkoss.zk.ui.Page | O | |
| | pageContext – org.zkoss.web.servlet.xel.PageContext | O | |
| | pageScope - java.util.Map | O | |
| | requestScope – java.util.Map | O | |
| | self - org.zkoss.zk.ui.Component | O | |
| | session - org.zkoss.zk.ui.Session | O | |
| | sessionScope - java.util.Map | O | |
| | spaceOwner - org.zkoss.zk.ui.IdSpace | O | |
| | spaceScope - java.util.Map | O | |
| Processing Instruction | The component Directive | O | |

| Topics | Items | | ZK JSP | Note |
|---|---|---|---|---|
| | attributes | class | O | |
| | | extend | O | |
| | | macroURI | O | |
| | | name | O | |
| | | moldURI | O | |
| | | moldName | O | |
| | The evaluator Directive | | X | |
| | attributes | class | X | |
| | | import | X | |
| | The forward Directive | | * | Not very useful because you can use <JSP:forward> instead |
| | attribute | uri | * | |
| | The import Directive | | * | Not very useful because you can use <JSP:include> to include a JSP page for setting purpose. |
| | attributes | uri | * | |
| | | directives | * | |
| | The init Directive | | O | |
| | attributes | class | * | Change to **use** attribute |
| | | zscript | O | |
| | | arg0, arg1... | O | |
| | The link and meta Directives | | * | Directly write them in JSP documents. |
| | attributes | href | * | |
| | | name0, name1... | * | |
| | The page Directive | | * | Implemented by tag: <zk:page> contentType,docType and xml could be written directly on JSP document.<br><br>The **complete** attribute is always false in JSP.<br><br>The **language** attribute is very rare to change but also implemented. |
| | attributes | cacheable | * | |
| | | complete | * | |
| | | contentType | * | |
| | | docType | * | |
| | | id | O | |
| | | language | O | |
| | | style | O | |
| | | title | O | |
| | | xml | * | |
| | | zscriptLanguage | O | |
| | The root-attributes Directive | | * | Directly write it on JSP Document |
| | attribute | any-name | * | |

| Topics | Items | | ZK JSP | Note |
|---|---|---|---|---|
| | The taglib Directive | | * | Use JSP <%@taglib%> instead |
| | attributes | uri | * | |
| | | prefix | * | |
| | The variable-resolver Directive | | O | |
| | attributes | class | * | Change to **use** attribute |
| | | arg0, arg1... | O | |
| | The xel-method Directive | | X | |
| ZK Elements | The XML Namespace | | * | Use JSP instead |
| | The attribute Element | | O | |
| | attributes | name | O | |
| | | trim | O | |
| | The custom-attributes Element | | O | |
| | attribute | scope | O | |
| | The variables Element | | O | |
| | attribute | local | O | |
| | The zk Element | | X | Useless in JSP |
| | The zscript Element | | O | |
| | attribute | language | O | |
| | | deferred | O | |
| | | src | O | |
| ZK Attributes | The apply Attribute | | O | |
| | The forEach Attribute | | * | Use JSTL **forEach** tag |
| | The forEachBegin Attribute | | * | Use <c:forEach begin> |
| | The forEachEnd Attribute | | * | Use <c:forEach end> |
| | The forward Attribute | | O | |
| | The fulfill Attribute | | X | JSP Specification Violation |
| | The if Attribute | | O | |
| | The unless Attribute | | O | |
| | The use Attribute | | O | |
| ZK Components | The XUL Components | | O | |
| | The XHTML Components | | * | Use HTML directly |