

# J-CASE



JSP Custom Taglib for Writing Storyboards

## **J-CASE Tag Library**

# **Storyboard Library Guide**

**Version 0.2.1**

Revised: December 2009

**Contents**

---

<b>1</b>	<b>Overview</b> .....	<b>3</b>
1.1	What is Storyboard Library? .....	3
1.2	Limitations .....	3
1.3	Coding Rules.....	4
1.4	Aspect Weaving .....	4
1.4.1	Page directive .....	5
1.4.2	J-CASE tag .....	6
1.4.3	Advice .....	6
1.5	Tools .....	8
1.5.1	Servlet Generator .....	8
1.5.2	Web.xml Updater .....	9
<b>2</b>	<b>Sample – Search Library</b> .....	<b>11</b>
2.1	STEP 1: Create Storyboard JSPs to be shared.....	11
2.2	STEP 2: Make a JAR file from the JSPs.....	14
2.3	STEP 3: Customize the base storyboard.....	15
<b>3</b>	<b>Sample - Authentication Library</b> .....	<b>19</b>
3.1	STEP 1: Create Storyboard JSPs to be shared.....	19
3.2	STEP 2: Compile the AUTH JSPs and make a auth-lib .....	21
3.3	STEP 3: Create SSO Storyboard JSPs .....	22
3.4	STEP 4: Compile the SSO JSPs and make a sso-lib .....	24
3.5	STEP 5: Create Application Storyboard JSPs .....	25
<b>4</b>	<b>Tips</b> .....	<b>28</b>
4.1	Regular Expression.....	28
4.2	Keep the Generated JSPs .....	29
4.3	How to find the class path and tag IDs .....	29
4.4	Weave Multiple Tags.....	30

## 1 Overview

---

This document is for system analysts or Java developers who create Storyboard Libraries to distribute and/or share the requirements with other projects.

### 1.1 What is Storyboard Library?

You may have seen similar use cases in some applications, for example, authentication, searching by keywords. The common use cases can be reused with other projects by creating a storyboard library.

J-CASE allows you to create a JAR file that can be distributed to other project by pre-compiling your storyboard JSPs.

### 1.2 Limitations

There are the following limitations with storyboard libraries:

- Only pre-compiled JSPs (servlet classes), jcase.xml, jcase.js and jcase.css can be included in the library.  
Other resources, such as image files, have to be distributed separately.
- Only the following JSP syntaxes are available. Others, e.g. JSP actions, JSTL, are not supported.
  - J-CASE taglib
  - page directive `<%@ page %>`
  - include directive `<%@ include %>`
  - declaration tag `<%! %>`
  - scriptlet tag `<% %>`
  - expression tag `<%= %>`
  - Comment `<%-- --%>`
  - Expression Language `${ }`
- All tags (J-CASE and HTML tags) must be closed properly.

```
Hello<br>
<input type="text" size="20">
<hr>
<input type="submit">
```

The above HTML tags have to be modified to below:

```
Hello<br/>
<input type="text" size="20"/>
<hr/>
<input type="submit"/>
```

- Cannot specify J-CASE tag in a HTML tag.

```
<body <c:if test="${flag}>onLoad="func()"</c:if>>
```

In this case, the above if tag can be specified in the func JavaScript function.

```
<html>
<head>
  <c:head />
  <script language="JavaScript" type="text/javascript">
    function func() {
      <c:if test="${flag}">
        location.href = location.href + "?generate=true";
      </c:if>
    }
  </script>
</head>
<body onload="func()">
...
```

### 1.3 Coding Rules

To make the library work on other storyboards, you need to follow the rules in your JSPs.

- Context path can't be specified in JSPs.  
The context path is not fixed when creating the library. It has to be specified using contextPath attribute in request object. If you want to describe the path below:

```
/root/UC01/index.jsp
```

The context path, '/root', should be modified to below:

```
${pageContext.request.contextPath}/UC01/index.jsp
```

The `${pageContext.request.contextPath}` is dynamically replaced with actual context path according to the environment.

Likewise, if you need to specify server name and port, serverName and serverPort API can be called.

```
http://localhost:8080/root/UC01/index.jsp
```

The above URL should be modified to below:

```
http://${pageContext.request.serverName}:${pageContext.request.serverPort}
${pageContext.request.contextPath}/UC01/index.jsp
```

### 1.4 Aspect Weaving

Even if we have similar requirements, the storyboard will be different depending on the project, because it describes detail specifications including UI, project specific requirements, etc. So, we cannot adopt the same storyboard library to all projects, and it has to have a capability to easily customize.

J-CASE enables to insert and replace with another J-CASE tag by aspect weaving. To weave aspects, you need to specify the following options in page directive or attributes in J-CASE tag.

### 1.4.1 Page directive

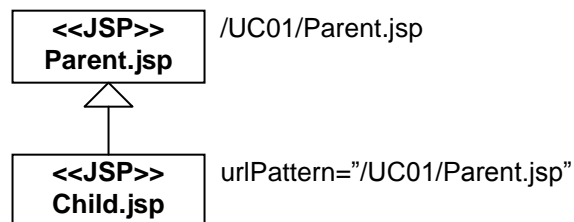
The following options in page directive are available to use with ServletGenerator tool:

- extends** : Extend the JSP or servlet class
- urlPattern** : Servlet URL pattern (optional)

For parentClass and parentId can be specified using regular expression to apply the change to multiple tags.

There are two choices to weave aspects, by extending JSP/class or specifying parent classes using parentClass attribute.

If you want to modify a specific JSP or class, e.g. Parent.jsp, you need to create another JSP named, for example, Child.jsp, which the Parent.jsp path in its “extends” attribute. Also, if the URL needs to be kept, you have to specify the URL pattern in urlPattern.



The J-CASE tags in the Child.jsp are refrected into Parent content. The page directive and J-CASE tag in the extended JSP are specified below:

```
<%@ page extends="/UC01/Parent.jsp" urlPattern="/UC01/Parent.jsp" %>
<c:tag
  id="message_new"
  parentId="message"
  advice="around">Company ID (<c:parent /></c:tag>
```

If the parent is a servlet class, you can specify the class path in extends.

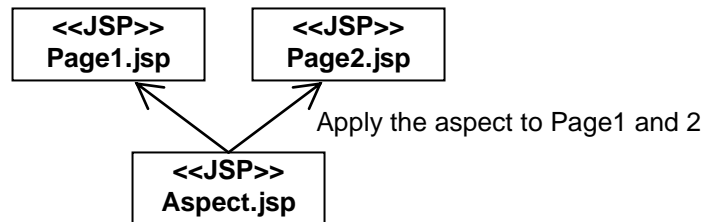
```
<%@ page extends="jcase.jsp.UC01.Parent_jsp" urlPattern="/UC01/Parent.jsp" %>
...
```

### 1.4.2 J-CASE tag

The following attributes in J-CASE tag are available to use with ServletGenerator tool:

- id** : Unique ID of the tag
- parentClass** : Parent JSP or class path (optional if the “extends” is specified)
- parentId** : Parent tag ID to be inserted or replaced
- advice** : Weaving type (before, after, around, top or bottom)

If you want to apply the aspect to multiple JSP/classes, you can specify parentClass attribute in any JSPs.



In this case, the parentClass attribute can be specified like below:

```

<c:tag
  id="message_new"
  parentClass="/UC01/Page[12].jsp$"
  parentId="message"
  advice="around">Company ID (<c:parent />)</c:tag>
  
```

If the target JSPs are provided as servlet classes, you need to specify the class path.

```

<c:tag
  id="message_new"
  parentClass="^jcase\\.jsp\\.UC01\\.Page[12]_jsp$"
  ...
  
```

To see the parent class path, you can access the page with your browser and open the source. You will find the servlet class path in a comment at the top of the HTML code.

```

<!-- J-CASE Servlet(jcase.jsp.UC01.Page1_jsp) -->

<html>
<head>
...
  
```

### 1.4.3 Advice

Advice attribute has to be specified in the J-CASE tag to be weaved to tell J-CASE where the tag should be inserted or replaced.

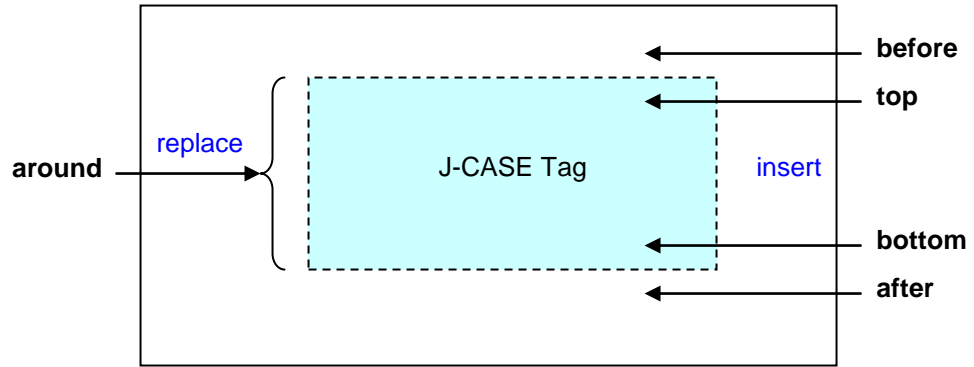
Before advice inserts the tag before the parent tag.

Top advice inserts it at inside top of the parent tag.

Bottom advice inserts it at inside bottom of the parent tag.

After advice inserts it after the parent tag.

Around advice replace the parent tag with the given tag. The parent tag can be displayed using <c:parent> tag.



## 1.5 Tools

J-CASE provides two tools; ServletGenerator for generating servlet source codes and web.xml fragment from JSPs and WebXmlUpdater to insert the web.xml fragments into the web.xml for the web project.

### 1.5.1 Servlet Generator

Once you create JSP files, you can generate the servlet source codes using ServletGenerator tool to make a JAR file (storybook library). It generates servlet container-neutral code so that you can deploy the JAR file on any containers. It can be invoked from Ant task as well as command line.

Parameter	Description	Required	Default
uriroot	Directory path that JSP files are stored.	Yes	--
inputFile	File paths to be processed	No	JSP files under uriroot
outputDir	Directory path that generated servlet codes are outputted.	No	./ (current directory)
webXmlFragment	Xml file name for web.xml fragment.	No	webXmlFragment.xml
encoding	Default encoding	No	Default encoding of the system
classpath	List of the class paths required for storyboard library. (Note: A path to servlet API jar has to be also specified)	No	--
keepGeneratedJSP	A flag to generate JSP files generated by this tool. <b>true</b> : generate <b>false</b> : not generated (default)	No	false

From command line, you can execute `jcasetool.ServletGenerator` class with appropriate parameters:

```
> java jcasetool.ServletGenerator -uriroot war -outputDir src -classpath "lib/servlet-api.jar, lib/jsp-api.jar, lib/jcasetool.jar"
```

If you use Ant, you can define a new task using `ServletGeneratorTask` class and call it in a task.

```
...
<path id="classpath">
  <fileset dir="lib">
    <include name="**/*.jar"/>
  </fileset>
</path>

<taskdef name="servletGenerator"
  classname="jcasetool.ServletGeneratorTask"
```

```

        classpath="lib/jcase.jar"/>

        <target name="generate">
            <servletGenerator
                uriroot="war"
                outputDir="src">
                <classpath refid="classpath"/>
            </servletGenerator>
        </target>
    ...

```

With the above parameters, you will find generated Java (and JSP) files in src directory after the execution.

As a default, all JSP files that the extension is “jsp” are processed. If you want to specific files or exclude some files, you can specify them using inputFile parameter.

The following code shows how to eliminate JSP files under “include” directory.

```

<target name="generate">
    <servletGenerator
        uriroot="war"
        outputDir="src">
        <inputFile>
            <fileset dir="war">
                <include name="**/*.jsp"/>
                <exclude name="**/include/**/*.jsp"/>
            </fileset>
        </inputFile>
    </servletGenerator>
</target>

```

### 1.5.2 Web.xml Updater

Storyboard classes generated and compiled by ServletGenerator tool provide web.xml fragments and you need to insert them into your web.xml using WebXmlUpdater tool.

It can be also invoked from Ant task as well as command line.

Parameter	Description	Required	Default
webXml	Path of web.xml for the project.	Yes	--
webXmlFragment	List of webXmlFragment.xml paths. If same definition (same servlet-name, url-pattern) exists in some XMLs, <b>the path specified earlier is effective.</b>	Yes	--
encoding	Default encoding of the web.xml specified in webXml parameter.	No	Default encoding of the system

From command line, you can execute jcase.tool.XmlWebUpdater class with appropriate parameters:

```

> java jcase.tool.WebXmlUpdater -webXml war/WEB-INF/web.xml -webXmlFragment
"lib1/webXmlFragment.xml, lib2/webXmlFragment.xml"

```

From Ant, you can define a new task using WebXmlUpdaterTask class to call the tool.

```

...
<path id="classpath">
  <fileset dir="lib">
    <include name="**/*.jar"/>
  </fileset>
</path>

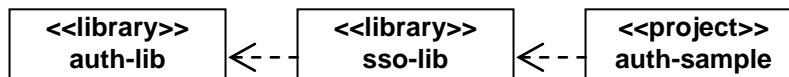
<taskdef name="webXmlUpdater"
  classname="jcase.tool.WebXmlUpdaterTask"
  classpath="lib/jcase.jar"/>

<target name="update">
  <webXmlUpdater webXml="war/WEB-INF/web.xml">
    <webXmlFragment>
      <pathelement location="lib1/webXmlFragment.xml"/>
      <pathelement location="lib2/webXmlFragment.xml"/>
    </webXmlFragment>
  </webXmlUpdater>
</target>
...

```

With the above parameters, servlet and servlet-mapping in fragment XMLs will be added in your war/WEB-INF/web.xml.

Note that the order of webXmlFragment XML files in the webXmlFragment tag is very important. The XML generated by the project which calls storyboard libraries must be described prior to the XMLs of the libraries. The entry found earlier is used, if there are two or more same entries. For example, there is a project named "auth-sample", which uses the "sso-lib" library. And the "sso-lib" uses the "auth-lib" library.



In this case, you must specify the XML paths with the following order:

1. webXmlFragment.xml for auth-sample project
2. webXmlFragment.xml for sso-lib
3. webXmlFragment.xml for auth-lib

```

...
<target name="update">
  <webXmlUpdater webXml="war/WEB-INF/web.xml">
    <webXmlFragment>
      <pathelement location="project/webXmlFragment.xml"/>
      <pathelement location="sso-lib/webXmlFragment.xml"/>
      <pathelement location="auth-lib/webXmlFragment.xml"/>
    </webXmlFragment>
  </webXmlUpdater>
</target>
...

```

If there is an entry with the same name in each XML, the entry defined in project/webXmlFragment.xml will be used.

## 2 Sample – Search Library

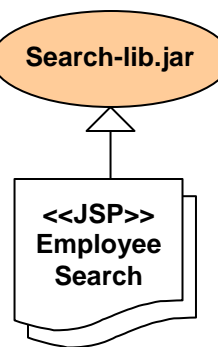
This chapter explains how to create a storyboard library of search use case and how to customize it using Aspect Weaving and Replacing String.

In this sample, first, you will create a regular storyboard with JSPs, which represent a simple use case only to search a keyword. And then you make a JAR file from the JSPs to distribute the use case to other projects. After that, you will create an Employ Search use case using the JAR file. This time, you don't have to create the use case from scratch. You just need to describe the application specific requirements by aspect weaving.

1. Create a base storyboard.



2. Compile the JSPs and make a JAR.



3. Create another storyboard by extending the base storyboard.

### 2.1 STEP 1: Create Storyboard JSPs to be shared

First, you need to create a storyboard JSPs which only describe common requirements so that you can share it with other projects. You will find the sample code in the following location:

```
<J-CASE HOME>/samples/search_lib/1_WAR/
```

This storyboard simply represents to search a keyword and show the results. So, the two JSPs are created:

- war/SRCH01/SearchForm.jsp: Search form page
- war/SRCH01/SearchResults.jsp: Search results page

The SearchForm.jsp describes a search form screen and the possible actions, entering a keyword and search for valid case and empty keyword for invalid. So, it has a <c:screen> tag for the search form and two <c:action> tag for the actions.

```
...
<body>
<center>

<c:title name="Search Form" /><br>

<c:screen id="screen_searchForm" name="Search Form" align="center">
  <table><tr>
    <td id="label_keyword">Keyword</td>
    <td id="textbox_keyword"><input size="30"/></td>
```

```

        </tr></table>

        <input id="button_search" type="button" value="Search" /><br/>
</c:screen>
<br/>
<c:actions>
    <c:action id="action_search"
        name="Search"
        link="SearchResults.jsp">
        The user enters a keyword and submits it.
    </c:action>

    <c:action id="action_empty"
        name="Empty keyword">
        The user submits the form with empty keyword.<br>
        The system displays an error message.
    </c:action>
</c:actions>
</center>
</body>
...

```

The code generates the screen below when executing on a servlet container.

Actions [hide]		
No.	Name	Description
1	Search	The user enters a keyword and submits it.
2	Empty keyword	The user submits the form with empty keyword. The system displays an error message.

To identify the J-CASE tags to may be weaved, id attribute is specified, e.g. screen\_searchForm for <c:screen> tag, action\_search and action\_empty for <c:action> tags.

The label "Keyword" for the text filed is also specified the id in HTML <td> tag using 'h' prefix. In this case, the <h:td> tag is marked as a joint point to be weaved an aspect as well as generating a regular <td> tag.

The SearchResult.jsp is also described with similar J-CASE tags with the search form. It has a search result screen and two actions with id attributes.

```

...
<body>

```

```
<center>

<c:title name="Search Results" /><br>

<c:screen id="screen_searchResults" name="Search Results" align="center">
  <table border="0">
    <tr><td>Result #1</td></tr>
    <tr><td>Result #2</td></tr>
    <tr><td>...</td></tr>
  </table>

  <u>Back</u><br>
</c:screen>
<br>

<c:actions>
  <c:action id="action_end"
    name="Confirm the results">
    The user gets the results related to the keyword.<br>
    The use case ends.
  </c:action>

  <c:action id="action_back"
    name="Back to the search form"
    link="SearchForm.jsp">
    There are no results expected.<br>
    The user goes back to the search form.
  </c:action>
</c:actions>

</center>
</body>
...
```

The code generates the screen below when executing on a servlet container.

**Search Results**

Screen: Search Results [hide]

Result #1  
Result #2  
...  
Back

Actions [hide]		
No.	Name	Description
1	Confirm the results	The user gets the results related to the keyword. The use case ends.
2	<a href="#">Back to the search form</a>	There are no results expected. The user goes back to the search form.

The use case name is defined in jcase.xml under the war/WEB-INF/classes/ so that JSP that calls this use case doesn't have to specify the use case name in <c"useCase> tag.

**war/WEB-INF/classes/jcase.xml**

```

...
    <useCases>
      <useCase id="SRCH01">Search</useCase>
    </useCases>
...

```

To make a WAR file, go to the project root and run Ant script:

```

> cd <J-CASE HOME>/samples/search_lib/1_WAR/
> ant

```

The WAR file, search-lib.war, will be generated in the same directory. Then run the ant script with deploy target to deploy the WAR onto Tomcat.

```

> ant deploy

```

Once it has been successfully deployed, start the Tomcat and access the URL below with your browser.

```

http://localhost:8080/search-lib/SRCH01/SearchForm.jsp

```

## 2.2 STEP 2: Make a JAR file from the JSPs

Next, you create a library (JAR file) from the JSPs created in previous step so that you can easily distribute the storyboard to other projects.

```
<J-CASE HOME>/samples/search_lib/2_LIB/
```

Go to the root directory for this step, and then run ant script with the f option.

```
> cd <J-CASE HOME>/samples/search_lib/2_LIB/
> ant -f build_servletGen.xml
```

The JSPs created in previous step will be compiled and put them into a JAR file. Also, it generates web.xml fragment file, which will be inserted into web.xml later.

- dist/search-lib.jar: storyboard library
- dist/webXmlFragment.xml: web.xml fragment

### 2.3 STEP 3: Customize the base storyboard

In this step, you want to define a use case to search an employee by a given employee ID. Since you have a search storyboard library, you don't have to define all. What you have to define is the application specific requirements. The sample code can be found in the following directory:

```
<J-CASE HOME>/samples/search_lib/3_EmployeeWAR/
```

The screen that you want to define in this step is below:

**[SRCH01] Search: Search Form**

Screen: Search Form
[\[hide\]](#)

[Help](#)

Employee ID

Please select an action from the following actions to go to the next page:

Actions <span style="float: right;"><a href="#">[hide]</a></span>		
No.	Name	Description
1	<a href="#">Search</a>	The user enters an employee ID and submits it.
2	<a href="#">Display help</a>	The user selects the help link to see search instructions.
3	Empty employee ID	The user submits the form with empty employee ID. The system displays an error message.

[Back](#)   [Main](#)

The elements appeared are almost same as the search storyboard defined in the first step. The differences are

- Help link is added in the screen to show search instructions.
- The "Display help" action is added for the Help link.

- The button name, “Search”, is changed to “Search Employee”.
- The text label, “Keyword”, in the screen is replaced with “Employee ID”
- The words, “keyword”, in the action table are replaced with “employee ID”.
- An instruction for actions is added before actions table.
- Back and Main links are added in the footer.

These changes are done in inherited JSP and aspect JSP. First, let’s take a look at inherited JSP, EmployeeSearchForm.jsp.

**war/UC01/EmployeeSearchForm.jsp**

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page extends="jcase.jsp.SRCH01.SearchForm_jsp"
    urlPattern="/SRCH01/SearchForm.jsp" %>

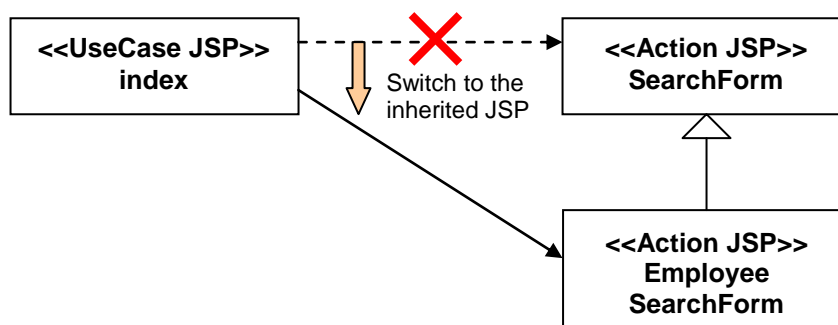
<c:tag id="link_help"
    parentId="screen_searchForm"
    advice="top">
    <p align="right" style="margin:0px;"><u>Help</u></p>
</c:tag>

<c:action id="action_help"
    parentId="action_search"
    advice="after"
    name="Display help"
    link="../UC01/SearchHelp.jsp">
    The user selects the help link to see search instructions.<br/>
</c:action>

<td id="label_search"
    parentId="label_keyword"
    advice="around">
    Employee ID
</td>

<input id="button_searchEmployee"
    parentId="button_search"
    advice="around"
    type="button"
    value="Search Employee"/>
```

As you can see the parent class path in “extends” attribute of the page directive, this is the JSP extended from the SearchForm.jsp in the storyboard JAR file. Also, urlPattern attribute shows that this JSP can be accessed using the same URL pattern as the parent JSP, “/SRCH01/SearchForm.jsp”.



With the “extends” and “urlPattern” attributes, the new JSP, EmployeeSearchFrom.jsp, inherits all contents from the SearchForm.jsp and all links pointing to the SearchForm.jsp are switched to access the new JSP.

In this JSP, four J-CASE tags are defined, which represent differences from the parent JSP.

1. Help link  
The first <c:tag> inserts Help link at the inside top of the screen tag.
2. Help action  
The second <c:action> tag inserts a new action for the Help link after action\_search action.
3. Employee ID text label  
The third <td> tag replaces the text field label with “Employee ID”.
4. Search button  
The last <input> tag replaces the button name with “Search Employee”.

These are not all for the changes listed above. The changes explained here are only applied on the JSP, SearchForm.jsp. But, the rest of changes should be applied on multiple pages, so you can defined them in an aspect JSP.

Aspect JSP is not special, it's just a JSP. You just need to specify parentClass attribute which represents JSPs and classes to be applied an aspect. So, technically, the aspect can be also defined in, for example, EmployeeSearchFrom.jsp, but you'll see aspects in aspect.jsp file in this sample for easy understanding.

**war/UC01/aspect.jsp**

```
<%@ page contentType="text/html;charset=UTF-8" %>

<c:inst id="actions_instruction"
  parentClass="*"
  parentId="^actions_tag_.*"
  advice="before">
  Please select an action from the following actions to go to the next page:
  <br>
</c:inst>

<c:tag id="footer"
  parentClass="*"
  parentId="^html_body_tag.*"
  advice="bottom">
  <c:include page="/UC01/footer.jsp"/>
</c:tag>

<c:replace parentClass="*" old="(\\W)a keyword(\\W)" new="$1an employee ID$2"/>
<c:replace parentClass="*" old="(\\W)keyword(\\W)" new="$1employee ID$2"/>
```

In the aspect.jsp, the four J-CASE tags are defined, and all parentClass attributes are specified with “.\*”, which means all JSPs and classes. So, the all changes are applied all JSPs and Servlet classes in library.

1. Insert action instruction  
The first <c:inst> tag inserts an instruction before all actions tags. All pages in this project and storyboard library don't specify id attribute for <c:actions> tag, J-CASE automatically assigns id starting with “actions\_tag\_”. The <c:inst> tag intends to change against all <c:actions> by specifying the regular expression “^actions\_tag\_.\*” (which means starting with “actions\_tag\_”) in parentId attribute.

2. Insert footer

The second `<c:tag>` represents to insert the application footer at the bottom of the body in every page.

3. Replace keyword

Two `<c:replace>` tags are intending to replace all the words, "keyword" with "employee ID". The `<c:replace>` tag replaces all words matched in the generated HTML, so if a link URL contains, for example, `"/Searchkeyword.jsp"`, the link is also replaced if you simply specify "keyword" in old attribute as the word to be replaced. Not to replace a part of the word, you can specify it, for example, `"(\\W)keyword(\\W)"`, which means "keyword" enclosed with special characters.

After creating the rest of JSPs (SearchHelp.jsp, footer.jsp, etc), go back to the root directory and run the ant script.

```
> cd <J-CASE HOME>/samples/search_lib/3_EmployeeWAR/  
> ant -f build_servletGen.xml
```

The script generates a storyboard library and a web.xml fragment file for this application under dist directory.

- dist/search-sample.jar: storyboard library
- dist/webXmlFragment.xml: web.xml fragment

Then, run build\_war.xml script with ant.

```
> ant -f build_war.xml deploy
```

It generates a WAR file, search-sample.war, in the same directory and updates web.xml with the web.xml fragment files. And then deploys the WAR on Tomcat.

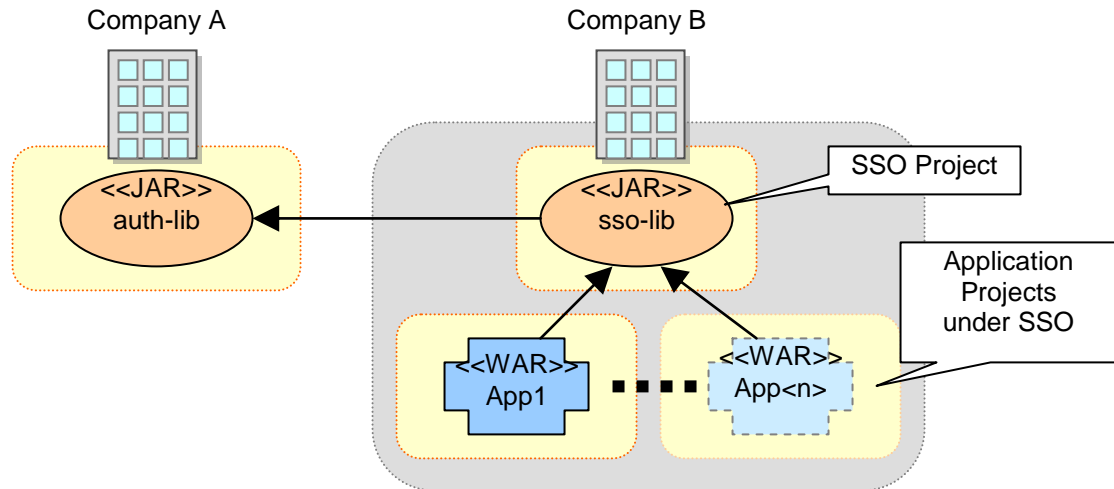
Once it has been successfully deployed, start the Tomcat and access the URL below with your browser.

```
http://localhost:8080/search-sample/
```

You'll see index page and the extended Employee Search form.

### 3 Sample - Authentication Library

This is a sample using two libraries. The first library is used as a prerequisite for the second library. In this sample, we assumed the following situation:



There are two companies, Company A and B. The company A develops and provides other companies an authentication component implementing AUTH01 and AUTH02 use cases. They also create storyboards of the use cases and put them into an auth-lib.jar file so that their customer can easily create own storyboard based on the library.

The company B is one of customers, which is planning to use the component in some projects. But, instead of simply applying it in each project, the company B decided to establish a new project, called SSO project, to make an own authentication component to enable single sign-on using the component of the company A. Then, the SSO project provides the new component for each sub projects working under the SSO.

The SSO project in Company B also generates a single sign-on storyboard using auth-lib.jar and makes another storyboard library, sso-lib.jar, which represents the SSO flows. The libraries are provided for sub projects and they're also creates own storyboard using the given SSO storyboard.

The steps to perform in this chapter are below:

- STEP 1: Company A: Create AUTH Storyboard JSPs
- STEP 2: Company A: Compile the AUTH JSPs and make a auth-lib
- STEP 3: Company B (SSO Proj): Create SSO Storyboard JSPs
- STEP 4: Company B (SSO Proj): Compile the SSO JSPs and make a sso-lib
- STEP 5: Company B (App Proj): Create Application Storyboard JSPs

#### 3.1 STEP 1: Create Storyboard JSPs to be shared

Company A creates storyboard JSPs to explain how their authentication component works. The JSPs and other required files can be found in the directory below:

<J-CASE HOME>/samples/auth\_lib/1\_CompanyA\_WAR/

This storyboard defines two use cases typically required for authentication:

- [COM\_A\_AUTH01] Log In
- [COM\_A\_AUTH02] Forgot Password

We assume that Use Case JSPs are created in each project, so the company A only creates Action JSPs. In this sample, they are the three JSPs below:

- **COM\_A\_AUTH01/LoginForm.jsp**: Login form to log in the system
- **COM\_A\_AUTH02/ForgotPassword.jsp**: Forgot Password form to ask security questions.
- **COM\_A\_AUTH02/ThankYou.jsp**: Thank You page to be shown when entering correct answers.

For example, the LoginForm JSP generates the following page including a login form and two actions.

[COM\_A\_AUTH01] Log In: Login Form

Screen: *Login Form*
[hide]

User ID

Password

Actions <span style="float: right;">[hide]</span>		
No.	Name	Description
1	Login	The user enters correct user ID and password, and logs in. The system creates session and navigate to the specified target page.
2	Invalid Credential	The user enters wrong user ID or password. The system displays an error message. (and rejoin this flow)

---

This is a sample footer for AUTH library. Please replace with your own footer.

The JSP codes are below:

**war/COM\_A\_AUTH01/LoginForm.jsp**

```

...
<c:screen id="screen_loginForm" name="<i>Login Form</i>" align="center">
<table border="0">
  <tr><td><c:tag id="userId_label">User ID</c:tag></td><td><input size="30"></td></tr>
  <tr><td><c:tag id="password_label">Password</c:tag></td><td><input
size="30"></td></tr>
</table>
<input type="button" value="Submit"><br>
</c:screen>
<br>

<c:actions>
  <c:action id="action_login"

```

```
name="Login"
useCaseId="{fn:directoryName(param.targetUrl, 1)}"

link="{param.targetUrl}?session=valid&${fn:excludesQueryString(pageContext,
'targetUrl, session')}">
    The user enters correct <c:tag id="userId_desc">user ID</c:tag> and
<c:tag id="password_desc">password</c:tag>, and logs in.<br>
    The system creates session and navigates to the specified target page.
</c:action>

    <c:action id="action_invalid" name="Invalid Credential">
    The user enters wrong <c:tag id="userId_desc">user ID</c:tag> or
<c:tag id="password_desc">password</c:tag>.<br>
    The system displays an error message. (and rejoin this flow)
    </c:action>
</c:actions>

<%@ include file="../../COM_A_include/footer.jsp" %>
```

**war/COM\_A\_include/footer.jsp**

```
<c:tag id="auth_footer">
    <br><hr>
    This is a sample footer for AUTH library.
    Please replace with your own footer.
</c:tag>
```

The most of the J-CASE tags are defined id attribute so that inherit / aspect JSPs can insert or replaced with a new tag.

In order to redirect to login page when the user accesses protected pages without logging in, this storyboard also provides JSP include file (authlib\_authenticate.jsp). It is supposed to be included in Action JSPs which represent a protected page. And the use case names are defined in jcase.xml for this library.

```
<useCases>
    <useCase id="COM_A_AUTH01">Log In</useCase>
    <useCase id="COM_A_AUTH02">Forgot Password</useCase>
</useCases>
```

Once you created all files, you can make a war file with the sample build.xml and deploy it on Tomcat to see if they work fine. The JSPs can be referred from the URLs below:

- [http://localhost:8080/auth-lib/COM\\_A\\_AUTH01/index.jsp](http://localhost:8080/auth-lib/COM_A_AUTH01/index.jsp)
- [http://localhost:8080/auth-lib/COM\\_A\\_AUTH02/index.jsp](http://localhost:8080/auth-lib/COM_A_AUTH02/index.jsp)

### 3.2 STEP 2: Compile the AUTH JSPs and make a auth-lib

Now that you've created all JSPs files for auth storyboard library, you can create a JAR file (storyboard library) from the JSPs. The ant script and properties files to create the library can be found in the following directory:

```
<J-CASE HOME>/samples/auth_lib/2_CompanyA_LIB/
```

In this step, you just need to execute the given ant task to generate a library.

```
> ant -f build_servletGen.xml
```

The task compiles the JSPs and generates a storyboard library (auth-lib.jar) file in dist directory. Also, in the same directory, you will find 'web\_xmlFragment.xml', which is a web.xml fragment file to be included in the web.xml of the project which uses the library.

Company A provides other companies the following files to share the storyboard:

- **authlib\_authenticate.jsp** (in 1\_CompanyA\_WAR)
- **auth-lib.jar** (in 2\_CompanyA\_LIB/dist)
- **webXmlFragment.xml** (in 2\_CompanyA\_LIB/dist)

### 3.3 STEP 3: Create SSO Storyboard JSPs

This is the step to be done by Company B that uses the auth-lib.jar. The company B gets the required files (auth-lib.jar, web.xml fragment file and include file) from Company A.

The SSO project team in Company B creates own single sign-on storyboards which access COM\_A\_AUTH01 and COM\_A\_AUTH02 use cases defined in the auth-lib library. The sample code for this step is in the following directory:

```
<J-CASE HOME>/samples/auth_lib/3_ComB_SSO_WAR/
```

The company B here creates common use cases required for sub projects. In this sample, you will add a use case, "[SSO\_UC01] Register User", which consists of the following two JSPs:

- **SSO\_UC01/UserReg.jsp**: User registration from
- **SSO\_UC01/ThankYou.jsp**: Thank you page for the user reg.

Next, they also need to customize the AUTH use cases in the auth-lib library to reflect the company specific requirements.

As you can see the following new login screen, the company B gets some requirements which are not implemented in the original auth-lib.

- Display a Forgot Password link

To implement it on the page, you can create a new JSP by extending from the LoginForm.jsp in the auth-lib.jar.

[COM\_A\_AUTH01] Log In: Login Form

SSO Login form

Screen: *Login Form* [hide]

Email (User ID)

Password

[Forgot Password](#)

(Extended by SSO library.)

No.	Name	Description
1	<a href="#">Login</a> (Use Case [UC01] Access Protected Page)	The user enters correct user ID and password, and logs in. The system creates session and navigates to the specified target page.
2	<a href="#">Forgot Password</a> (Use Case [COM_A_AUTH02] Forgot Password)	The user selects "Forgot Password" to see own password.
3	Wrong ID or password	The user enters wrong user ID or password.

---

This is a sample footer for SSO library.

The new login JSP for SSO project, SsoLoginForm.jsp, can be found in war/SSO\_AUTH01. Below is a part of the JSP code, which shows what tags are overridden.

**war/SSO\_AUTH01/SsoLoginForm.jsp**

```

<%@ page extends="jcase.jsp.COM_A_AUTH01.LoginForm_jsp" %>
...
<c:tag id="screen_loginForm_ssoLinks"
  parentId="screen_loginForm"
  advice="bottom">
  <u>Forgot Password</u><br>
</c:tag>

<c:action id="action_forgotPwd"
  parentId="action_login"
  advice="after"
  name="Forgot Password"
  useCaseId="COM_A_AUTH02"
  link="../COM_A_AUTH02/ForgotPassword.jsp">
  The user selects "Forgot Password" to see own password.
</c:action>
...

```

The page directive at the first line specifies that this extends from LoginForm.jsp class. So, only J-CASE tags to be overridden can be defined in this JSP.

First, in order to show the Forgot Password link in the screen, the <c:tag> with parentId=screen\_loginForm is defined. According to the advice=bottom, the tag is inserted at inside bottom of the screen tag in parent class.

Next, you also need to add the action for the link in actions table. So, the <c:action> tag with parentId=action\_login is specified. Since the advice is "after", this action is appended after the parent action.

You have one more change that the footer should be replaced with SSO footer. However, it's not only for this page but also for all pages in auth-lib. So, you can weave the change by specifying parentClass attribute.

#### **war/SSO\_aspect/aspect.jsp**

```
<%@ page contentType="text/html;charset=UTF-8" %>
<%@ taglib prefix="c" uri="http://www5f.biglobe.ne.jp/~webtest/jcase/core" %>

<%-- Replace auth footer in AUTH packages with SSO footer --%>
<c:tag id="sso_footer"
      parentClass=".*_AUTH.*"
      parentId="auth_footer"
      advice="around">
    <c:include page="/SSO_include/footer.jsp"/>
</c:tag>
```

The <c:tag> tag is applied to all classes containing "\_AUTH" in its class path. The J-CASE tags which id is "auth\_footer" are replaced with the tag.

Once you create all files, you can make a war file with the sample build.xml and deploy it on Tomcat. The JSPs can be referred from the URLs below:

- [http://localhost:8080/sso-lib/SSO\\_UC01/index.jsp](http://localhost:8080/sso-lib/SSO_UC01/index.jsp)
- [http://localhost:8080/sso-lib/SSO\\_AUTH01/index.jsp](http://localhost:8080/sso-lib/SSO_AUTH01/index.jsp)

### **3.4 STEP 4: Compile the SSO JSPs and make a sso-lib**

This step makes a sso-lib.jar by compiling the JSPs created in previous step. The ant script and properties files to create the library can be found in the following directory:

```
<J-CASE HOME>/samples/auth_lib/4_ComB_SSO_LIB/
```

In this step, you just need to execute the given ant task to generate a library.

```
> ant -f build_servletGen.xml
```

The task compiles the JSPs and generates a storyboard library (sso-lib.jar) file in dist directory. Also, in the same directory, you will find 'web\_xmlFragment.xml', which is a web.xml fragment file to be included in the web.xml of the project which uses the library.

SSO project team provides sub project teams in the company the following files:

- **sso-lib.jar** (in 4\_ComB\_SSO\_LIB/dist)
- **webXmlFragment.xml** (in 4\_ComB\_SSO\_LIB/dist)

### 3.5 STEP 5: Create Application Storyboard JSPs

This is the final step to be done by application team, which will develop an application using the components provided by Company A and SSO project of Company B. The sample code for this step is in the following directory:

```
<J-CASE HOME>/samples/auth_lib/5_ComB_App_WAR/
```

The application team here creates the application specific use cases. In this sample, you will add a use case, "[UC01] Register User", which consists of the following two JSPs:

- **UC01/Session.jsp**: To check to see if the request has a valid session.
- **UC01/Target.jsp**: Protected page by the session.

Next, they also need to customize the SSO use cases in the sso-lib library to reflect the application specific requirements, which is to show a Create Account option on the login page.

[COM\_A\_AUTH01] Log In: Login Form

SSO Login form

Screen: *Login Form*
[hide]

My ID [ Email (User ID) ]

Password

[Forgot Password](#)

[Create Account](#)

For detail screen, see [Login Screen](#).

(Extended by SSO library.)

Actions <span style="float: right;">[hide]</span>		
No.	Name	Description
1	<a href="#">Login</a> (Use Case [UC01] Access Protected Page)	The user enters correct user ID and password, and logs in. The system creates session and navigates to the specified target page.
2	<a href="#">Forgot Password</a> (Use Case [COM_A_AUTH02] Forgot Password)	The user selects "Forgot Password" to see own password.
3	<a href="#">Create Account</a> (Use Case [SSO_UC01] Register User)	The user selects "Create Account" to create a new account.
4	Wrong ID or password	The user enters wrong user ID or password.

BACK    MAIN-MENU

The new login JSP for this application, AppLoginForm.jsp, can be found in war/AUTH01. Below is a part of the JSP code, which shows what tags are overridden.

**war/AUTH01/AppLoginForm.jsp**

```
<%@ page extends="jcase.jsp.SSO_AUTH01.SsoLoginForm_jcase_ex_1_jsp"
    urlPattern="/COM_A_AUTH01/LoginForm.jsp" %>

<c:tag id="screen_loginForm_appLinks"
    parentId="screen_loginForm"
    advice="bottom">
    <u>Create Account</u><br>
    <br>
    For detail screen, see
    <c:link name="loginScreen" href="/auth-sample/screens.jsp" target="_blank">
        Login Screen.
    </c:link>
</c:tag>

<c:action id="action_createAccount"
    parentId="action_forgotPwd"
    advice="after"
    name="Create Account"
    useCaseId="SSO_UC01"
    link="../SSO_UC01/UserReg.jsp">
    The user selects "Create Account" to create a new account.
</c:action>
...
```

The page directive at the first line specifies that this extends from SsoLoginForm.jsp class. Also, urlPattern is specified to use the original login page URL. This JSP only allows defining J-CASE tags to be overridden due to extended JSP.

First, in order to show the Create Account link in the screen, the <c:tag> with parentId=screen\_loginForm\_appLinks is defined. According to the advice=bottom, the tag is inserted at inside bottom of the screen tag in parent class. So, the new link will be appeared under Forgot Password link.

Next, you also need to add the action for the link in actions table. So, the <c:action> tag with parentId=action\_forgotPwd is specified. Since the advice is "after", this action is appended after the Forgot Password action.

You have one more change that the footer should be replaced with the application specific one. The change is not only for this page but also for all pages in sso-lib. So, you can weave the change by specifying parentClass attribute.

**war/aspect/aspect.jsp**

```
<%@ contentType="text/html;charset=UTF-8" %>
<%@ taglib prefix="c" uri="http://www5f.biglobe.ne.jp/~webtest/jcase/core" %>

<%-- Replace SSO footer in SSO packages with own footer --%>
<c:tag id="my_footer"
    parentClass=".*SSO_.*"
    parentId="sso_footer"
    advice="around">
    <c:include page="/include/footer.jsp"/>
</c:tag>
```

The <c:tag> tag is applied to all classes containing “SSO\_” in its class path. The J-CASE tags which id is “sso\_footer” are replaced with the tag.

After creating the rest of JSPs, go back to the root directory and run the ant script.

```
> cd <J-CASE HOME>/samples/auth_lib/5_Comb_App_WAR/  
> ant -f build_servletGen.xml
```

The script generates a library and a web.xml fragment file for this application under dist directory.

- dist/auth-sample.jar: storyboard library
- dist/webXmlFragment.xml: web.xml fragment

Then, run build\_war.xml script with ant.

```
> ant -f build_war.xml deploy
```

It generates a WAR file, auth-sample.war, in the same directory and updates web.xml with the web.xml fragment files. And then deploys the WAR on Tomcat.

Once it has been successfully deployed, start the Tomcat and access the URL below with your browser.

```
http://localhost:8080/auth-sample/
```

You'll see index page and the application specific use cases extended from SSO and AUTH libraries.

## 4 Tips

This chapter shows some tips for creating storyboard library.

### 4.1 Regular Expression

The parentId and parentClass attributes accept regular expression to specify id and JSP/class path to be weaved. This section shows typical expressions that most developers may use.

Condition	Regular Expression	Sample Strings	
		Matched	Not matched
Matches all IDs, JSP/class paths	.*	<Any strings>	<N/A>
Starts with 'action'	^action.*	action actions action_tag	my_action
Ends with 'jsp'	.*jsp\$	Test.jsp jcase.jsp.Test_jsp	Test.jspx
Starts with 'action' or 'screen'	^(action screen).*	action screen action_tag screen_tag	my_action my_screen
Contains 'action'	.*(action).* or (?=.*action).*	my_action_tag actions youraction	
Contains 'action' or 'screen'	.*(action screen).* or (?=.*(action screen)).*	my_action_tag my_screen_tag	
Do not start with 'action'	(?!action).*	my_action	action action_tag
Do not start with 'action' or 'screen'	(?!(action screen)).*	my_action my_screen	action screen action_tag screen_tag
Do not contain 'action'	(?!.*action).*		action action_tag my_action
Do not contain 'action' and 'screen'	(?!.*(action screen)).*		action screen action_tag screen_tag my_action my_screen
Contains 'action' but do not contain 'screen'	(?=.*action)(?!.*screen).*	action my_action_tag	action_screen action screen

Also a dot is used to express any character, so a dot as a character needs to be escaped using a backslash. If you want to express a JSP or class path, for example,

jcase.jsp.Test.jsp
--------------------

you need to specify below:

```
parentClass="jcase\.jsp\.Test\.jsp"
```

If you don't want to contain JSPs/classes under 'include' directory/package, you can describe below:

```
parentClass="(?!.*include).*"
```

If you don't want to specify ID that expresses to start 'head\_tag' but 'head\_tag\_jcase', you can describe below:

```
parentId="(?!head_tag)(?!head_tag_jcase).*"
```

## 4.2 Keep the Generated JSPs

ServletGenerator tool generates a JSP file from the original JSP(s) before generating Servlet (Java) code. If you want see the JSP files for debugging purpose, you can specify keepGeneratedJSP option with the value 'true'.

```
<target name="generate">
  <servletGenerator
    uriroot="war"
    outputDir="src"
    keepGeneratedJSP="true"/>
</target>
```

After executing the task, you will find JSP files under the uriroot directory. In case of a regular JSP, you will see the JSP that J-CASE tags have been converted into Servlet code. In case of aspect JSPs, you will find a JSP and backup file (the extension is jsp\_bak). The backup file is the JSP the all aspect tags for the class are described. And the JSP is the one converted from the backed up JSP like the regular JSP case.

## 4.3 How to find the class path and tag IDs

To weave aspects, you need to know the JSP class path and tag ID of the target tag. You will be able to see the ID if you have the original JSP file. However, it's usually provided as a JAR file after compiling. Also, the JSP (Servlet) class might be inherited by Aspect JSPs. You can find the IDs with the following ways:

### HTML Source Code

J-CASE outputs the Servlet class path and ID of each tag in generated HTML. To see the information, first you need to deploy the library and access the page with your browser, and then view the HTML source. You'll see the Servlet class path in the first HTML comment, HTML ID in id attribute of the HTML tag and J-CASE ID in other HTML comments.

```
<!-- J-CASE Servlet(jcase.jsp.Test_jsp) -->
<html id="html_html_tag_jcase_jsp_Test_jsp_1">
<head id="html_head_tag_jcase_jsp_Test_jsp_2">
...
</head>
```

JSP class path

ID for HTML tag

```

<body id="body">
<center id="html_center_tag_jcase_jsp_Test_jsp_6">

<!-- J-CASE Tag(TitleTag) ID(title_tag_jcase_jsp_Test_jsp_7) -->
<jcase:data jcase_id="TitleTag" jcase_name="Test" />
<table border="1" width="100%" cellspacing="0" cellpadding="5" bgcolor="white">
<tr><td align="center"><span class="jcase_size6"><strong><font color="black">[UC01]
Test: Page1</font></strong></span><br></td></tr>
</table>
<br id="html_br_tag_jcase_jsp_Test_jsp_8"/>

<!-- J-CASE Tag(ScreenTag) ID(screen_testPage) -->
<jcase:data jcase_id="ScreenTag" jcase_name="Test Page">

<table border="0" width="" cellspacing="0" cellpadding="0">
<tr><td>
...

```

If the ID is specified in JSP, you will see the ID there. If not, the ID generated by J-CASE is set. It is generated with "html" (if HTML) + tag name + "tag" + class path + sequential number.

**Tooltip**

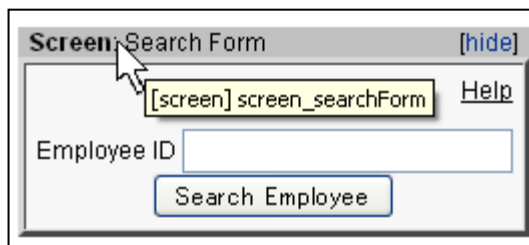
If you change "showTagId" system attribute to true, J-CASE pops a tooltip with the tag ID. Open jcase.xml file in your project and add the following node. Then restart the web application.

```

jcase.xml
<system>
  <attribute name="showTagId">true</attribute>
</system>

```

When you move your mouse to an element generated by J-CASE or HTML tag, you will see the Tag ID in the popped-up tooltip.



**4.4 Weave Multiple Tags**

The target tag in parent JSP or class is allowed to be weaved only a tag at a time. If you need to, for example, insert two action tags, those tags have to be enclosed in another tag. In this case, <c:tag>, which does output nothing, is used for this purpose. For example, there is the action with the id=action1 in a parent JSP.

```

<c:actions>
  <c:action id="action1"
    name="Action 1">
    The user selects the action 1.
  </c:action>
</c:actions>

```

```
...  
</c:actions>
```

If you want to add action2 and 3 after the action1, you can describe the tag below in a child JSP or aspect JSP:

```
<c:tag id="group1"  
  parentClass="*" *"  
  parentId="action1"  
  advice="after">  
  <c:action id="action2"  
    name="Action 2">  
    The user selects the action 2.  
  </c:action>  
  <c:action id="action3"  
    name="Action 3">  
    The user selects the action 3.  
  </c:action>  
</c:tag>
```