

J-CASE



JSP Custom Taglib for Writing Storyboards

J-CASE Tag Library

User's Guide

Version 0.2.1

Revised: December 2009

Contents

1	Overview	3
1.1	What is J-CASE?	3
2	Installation	3
2.1	Prerequisites	3
2.2	Downloading and deploying the JARs	3
3	Getting Started	4
3.1	Download sample WAR	4
3.2	Write a use case	5
3.3	Represent the use case on JSPs.....	5
3.4	Generate use case document from the JSPs	7
3.5	Write storyboard based on the use case JSPs	7
3.6	Generate the storyboard document	9
3.7	Refine the storyboard.....	10
4	Process Methods	14
5	Custom Style sheet	16
6	User JavaScript	16
7	jcse.xml	17
7.1	System Attributes	17
7.2	Tag Attributes	17
7.3	User Variables.....	18
7.4	Use Case Names	19
7.5	Actor Names	20
8	Generate Use Case Document	21
8.1	<c:generateUseCase> tag	21
8.2	UseCaseGenerator command	22
8.3	UML Diagrams	24
9	Basic Authentication	27
10	How to use JSTL	28
11	Tips	29
11.1	Print background colors and images.....	29
11.2	Convert the Use Case document into MS-WORD.....	30
11.3	Authentication	31
11.4	URL Parameters to be ignored in generateUseCase	32

1 Overview

This document is for system analysts who are starting to develop Use Cases and/or Storyboards using J-CASE tag library.

1.1 What is J-CASE?

J-CASE is a tag library for writing Use Cases and Storyboards. Each events and flows appear in use cases are described on JSP. It works as a sort of prototype on servlet container such as Tomcat so that stakeholders can experience the requirements by clicking links. Also regular Use Case / Storyboard documents can be generated from the JSP.

2 Installation

Installation of J-CASE is quite easy. You just need to download JAR files and put it into lib directory in your site.

2.1 Prerequisites

Before starting to install J-CASE, you need to install or download the following software:

- **Java SE** Version 5.x or 6.x
- **Apache Tomcat** Version 5.5.x or 6.0.x
It must also work on other servlet containers that implement the Servlet 2.4 and JavaServer Pages 2.0, but we tested only on the Tomcat.
- **Apache Ant** Version 1.7.0
This is required to make a build.
- [OPTIONAL] **UMLet** Ver. 10.3
<http://www.umlet.com/>
If you want to edit Use Case and Navigation Map diagrams, the UMLet is required. (See 'UML Diagrams' in 'Generate Use Case Document' chapter)
- [OPTIONAL] **HSQLDB** Version 1.8.0.8 (hsqldb.jar)
<http://hsqldb.org/>
If you use <c:feedback> tag, this JAR is required.
The JAR can be also found in our sample war file (e.g. cal-sample.war).
- [OPTIONAL] **Jakarta Standard Taglib (JSTL)** Version 1.1.2
<http://jakarta.apache.org/taglibs/> (jstl.jar and standard.jar)
NOTE: JSTL is not supported in JSPs for ServletGenerator.
If you want to use additional tags and/or functions, see 'How to use JSTL' chapter.

2.2 Downloading and deploying the JARs

Download the latest version of J-CASE (jcase.jar) from the site below:

<http://www5f.beglobe.ne.jp/~webtest/jcase/>

Then, deploy it with other required JARs into lib directory in your site.

Or you can easily start with a sample war file, `login-sample.war`, including required JAR files. (See Getting Started)

3 Getting Started

To demonstrate writing a use case and the storyboard with J-CASE, we use a sample use case "Log in" (`login-sample.war`) that is available to download on the J-CASE web site. The steps to do are as follows:

1. Download sample WAR
2. Write a use case
3. Represent the use case on JSPs
4. Generate use case document from the JSPs
5. Write storyboard based on the use case JSPs
6. Generate the storyboard document
7. Refine the storyboard

3.1 Download sample WAR

Before writing use cases and storyboards, take a look at how the sample works. The steps to download and deploy are:

1. Download **login-sample.war** file from the J-CASE site below:

<http://www5f.beglobe.ne.jp/~webtest/jcase/>

2. Copy the `login-sample.war` in `<TOMCAT_HOME>/webapps` directory.
(e.g. `C:\apache-tomcat-5.5.x\webapps`)
3. Start the Tomcat
4. Open your browser and access **`http://localhost:8080/login-sample/`**
You will see the page showing links to Use Case and Storyboard.

This sample shows three types of use cases; use case, storyboard with screens and storyboard with screens and processes.

3.2 Write a use case

First, we must focus on writing use cases with gathering information on what business users want to do. Here we start with the use case "Log in" shown in Figure 1.

Figure 1. "Log in" Use Case Description.

ID	UC01
Name	Log in
Brief Description	The user logs in the system.
Actor	Consumer user

Main Flow

1. This use case starts when the user accesses a protected page. The system displays a login form.
2. The user enters ID and password. The system validates them, and displays the protected page the user requested. The use case ends.

Alternative Flow 1 Starts after Main Flow - step 1

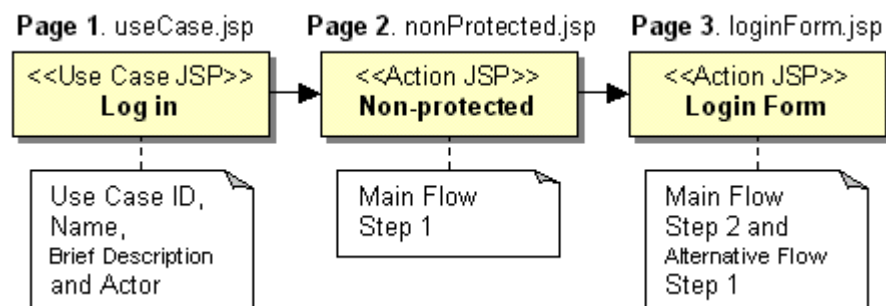
1. The user enters invalid ID or password. The system displays an error message. The use case ends.

In this use case, there are two flows, main and alternative flows. In the main flow, the user is authenticated by entering correct ID and password on login form, and will see the protected page. On the other hand, the alternative flow represents an error case that the user enters a wrong ID or password.

3.3 Represent the use case on JSPs

In this step, we will create JSP pages that represent the use case using J-CASE tags. Figure 2 shows three JSPs to be created here.

Figure 2. JSPs to be created for "Log in" use case.



In the first JSP, useCase.jsp, we describe use case ID, name, brief description and actor using <c:useCase> and <c:desc> tags.

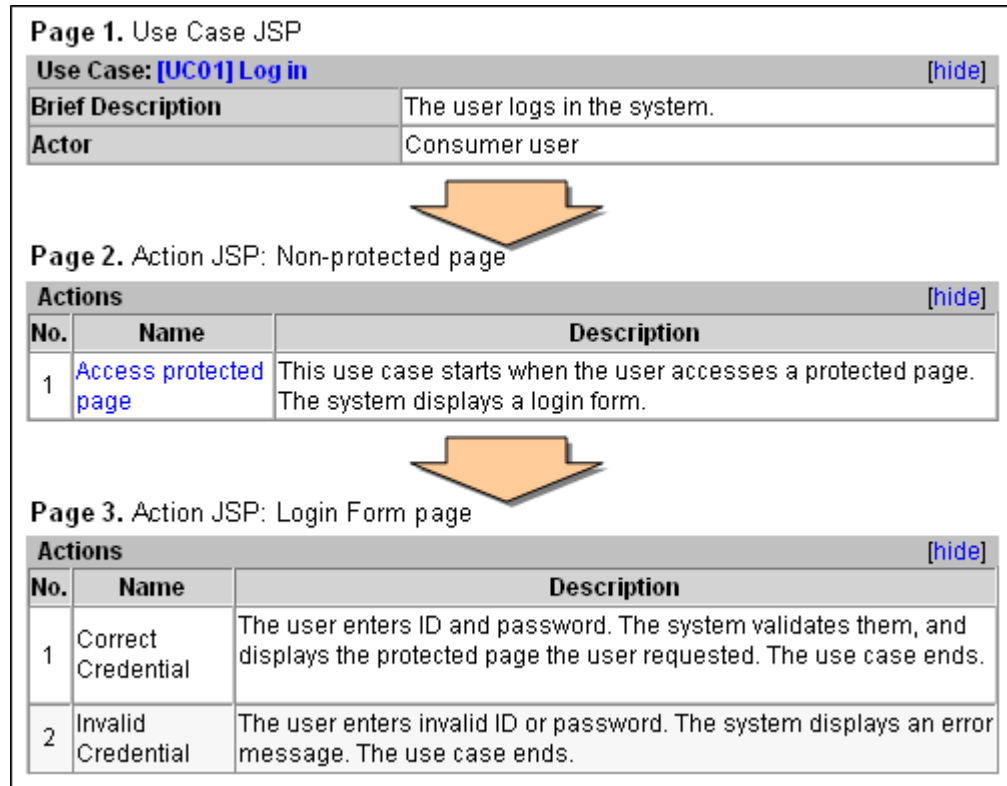
Listing 1. useCase tag in useCase.jsp.

```
<c:useCase id="UC01" name="Log in" link="nonProtected.jsp">
  <c:desc name="Brief Description">The user logs in the system.</c:desc>
  <c:desc name="Actor"> Consumer user</c:desc>
```

```
</c:useCase>
```

The <c:useCase> tag generates a use case table shown at Page 1 in Figure 3.

Figure 3. "Log in" Use Case represented in JSPs.



The table displays a link labeled "[UC01] Log in" that is linked to the next page, nonProtected.jsp. In this JSP, the flows of events are not described. They are described in the second and third JSPs.

The second JSP, nonProtected.jsp, represents a non-protected page described at step 1 in Main Flow in Figure 2. Here, we need to identify the user and/or system actions that can be performed on this page. In this case, we can describe the description of the Main Flow step 1 into <c:action> tag as shown in Listing 2.

Listing 2. Action tag in nonProtected.jsp.

```
<c:actions>
  <c:action name="Access protected page" link="loginForm.jsp">
    This use case starts when the user accesses a protected page.
    The system displays a login form.
  </c:action>
</c:actions>
```

The <c:actions> tag generates an actions table showing the action name and description shown at Page 2 in Figure 3. The action is linked to the URL specified in the link attribute. Since the protected page requires authentication and the user is not logged in the system yet, the next page to be displayed is login page. That is the reason that "loginForm.jsp" is specified in the link attribute.

The third JSP to be created is the loginForm.jsp that represents a login form. According to the “Log in” use case, there are two actions. One is the action that the user enters correct ID and password (at step 2 in Main Flow). The other is the action that the user enters wrong ID or password (at step 1 in Alternative Flow). So, two actions are described here as you can see in Listing 3.

Listing 3. Action tag in loginForm.jsp.

```
<c:actions>
  <c:action name="Correct Credential">
    The user enters ID and password.
    The system validates them,
    and displays the protected page the user requested.
    The use case ends.
  </c:action>
  <c:action name="Invalid Credential">
    The user enters invalid ID or password.
    The system displays an error message.
    The use case ends.
  </c:action>
</c:actions>
```

Since the both flows end here, link attribute in each <c:action> is not specified. This JSP generates the actions table containing two rows as shown at Page 3 in Figure 3. Now that all JSPs for “Log in” use case were created. Once the JSPs are deployed on servlet container, we can experience the flows by clicking the links (Page 1 through 3 in Figure 3).

3.4 Generate use case document from the JSPs

J-CASE allows us to generate use case document from the JSPs created in the previous step. To generate the document, <c:generateUseCase> tag can be used.

Listing 4. Generate a Use Case document.

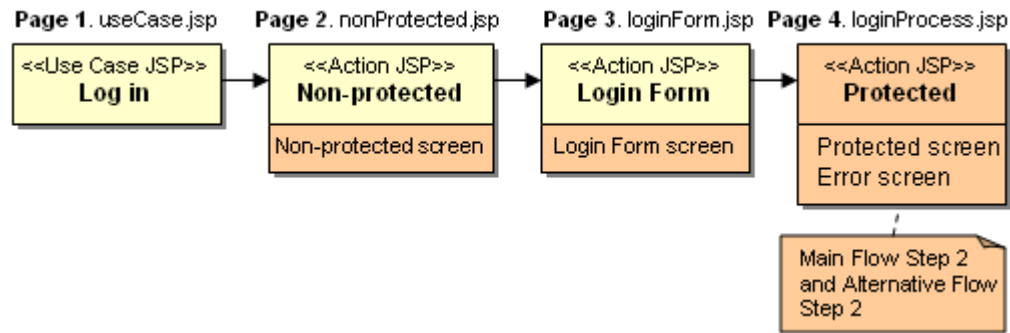
```
<c:generateUseCase
  input="useCase.jsp"
  output="useCase.html"
  showName="false"
  showScreen="false"
  showProcessResults="false"
/>
```

The tag requires input and output attributes. We need to specify the path of useCase.jsp in the input attribute and a path of use case HTML to be generated in the output attribute. Also, we can specify some more attributes such as showName in order to generate it with better style. Refer to the use case in Figure 1 again. It is the use case HTML generated by the <c:generateUseCase> tag.

3.5 Write storyboard based on the use case JSPs

We have captured high-level requirements in JSPs and the use case document has been generated. Now, it is time to create storyboard that describes how the user interacts with the system by adding abstract screens and system processes. In this step, we will create a new JSP, protected.jsp, with two screens, and we add two more screens into nonProtected.jsp and loginForm.jsp as shown in Figure 4.

Figure 4. JSPs to be created for “Log in” storyboard.



The first JSP, useCase.jsp, has no changes. So, we use the same code for this storyboard as well.

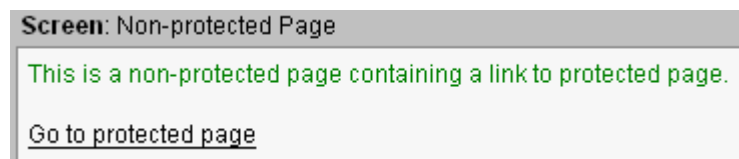
In the second JSP, nonProtected.jsp, <c:screen> tag that represents the Non-protected page is added. As shown in Listing 5, the <c:screen> tag contains conceptual content that simply displays a message with underline representing a link to protected page.

Listing 5. Screen tag in nonProtected.jsp.

```
<c:screen name="Non-protected Page">
  <c:inst>This is a non-protected page containing a link to protected page.</c:inst>
  <br><br>
  <u>Go to protected page</u>
</c:screen>
```

When the JSP is executed, the <c:screen> tag is converted into the screen shown in Figure 5.

Figure 5. Screen frame of non-protected page.



In the third JSP, loginForm.jsp, we add <c:screen> tag that represents a login form. Also, we add a link attribute in each <c:action> tag to make a link to the next JSP, protected.jsp with result parameter. The parameter tells the protected.jsp whether or not credential is correct.

Listing 6. Screen and Actions in login Form.jsp.

```
<c:screen name="Login Form" align="center">
  <table>
    <tr><td>User ID</td><td><input type="text"></td></tr>
    <tr><td>Password</td><td><input type="text"></td></tr>
  </table>
  <input type="button" value="Submit">
</c:screen>
<br>
<c:actions>
  <c:action name="Correct Credencial"
```

```

        link="protected.jsp?result=success">
        The user enters ID and password.
    </c:action>
    <c:action name="Invalid Credencial"
        link="protected.jsp?result=failure">
        The user enters invalid ID or password.
    </c:action>
</c:actions>

```

In the last JSP, protected.jsp, two results are described separately, and one of them is displayed depending on the value of the result parameter passed by loginForm.jsp.

Listing 7. Two results in protected.jsp.

```

<c:choose>
  <c:when test="{param.result == 'success'}">
    <c:screen name="Protected Page">
      <c:inst>Display contents the user requested.</c:inst>
    </c:screen>
    <br>
    <c:actions>
      <c:action name="Display the protected page">
        The system authenticates the user,
        and displays the protected page requested.
        The use case ends.
      </c:action>
    </c:actions>
  </c:when>
  <c:otherwise>
    <c:screen name="Error page" bgcolor="pink">
      <c:inst>Display an error message.</c:inst>
    </c:screen>
    <br>
    <c:actions>
      <c:action name="Login Error">
        The system cannot find the user information
        with the ID and password.
        The system displays an error message.
        The use case ends.
      </c:action>
    </c:actions>
  </c:otherwise>
</c:choose>

```

If the “param.result” is "success", the protected content represented by the <c:screen> and <c:actions> in <c:when> tag is displayed. Otherwise, the error content in <c:otherwise> tag is displayed.

3.6 Generate the storyboard document

In this step, we generate storyboard document from the modified JSPs using the same tag, <c:generateUseCase>, which we have used for the use case document. Since we want to show all information including screens and process results, we don't specify the showName, showScreen and showProcessResults attributes this time (the default values for the attributes are 'true').

Listing 8. Generate a Storyboard document

```
<c:generateUseCase
    input="useCase.jsp"
    output="storyboard.html"
/>
```

The <c:generateUseCase> generates the HTML shown in Figure 6.

Figure 6. Generated storyboard document (Main Flow only).

ID	UC01
Name	Log in
Brief Description	The user logs in the system.
Actor	Consumer user

Main Flow

1. Access protected page
This use case starts when the user accesses a protected page. The system displays a login form.

Screen: Non-protected Page

This is a non-protected page containing a link to protected page.

[Go to protected page](#)

2. Correct Credencial
The user enters ID and password.

Screen: Login Form

User ID

Password

3. Display the protected page
The system authenticates the user, and displays the protected page requested. The use case ends.

Screen: Protected Page

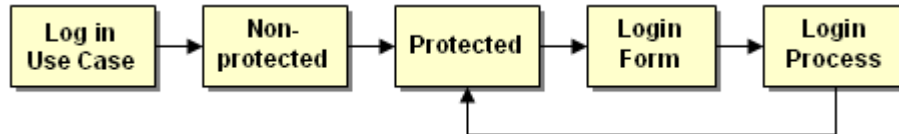
Display contents the user requested.

3.7 Refine the storyboard

The storyboard created in step 4 treats the system as a black box, so system processes are not mentioned. It should okay for discussions with business users, however, if you want to

describe more realistic flows, system processes should be also considered. For example, in real world, the non-protected.jsp doesn't link to login page but protected page. In order to refuse requests without coming through login page, all protected pages have to check if the user has already logged in by validating the session or something. In this step, we will add authentication processes to express detail login logic. Therefore, the original page flow is changed to the new flow shown in Figure 7.

Figure 7. Revised JSP flow.



At the first attempt to access the protected page from non-protected, it redirects to login form page because of no session created. After the user enters correct ID and password, the content of protected page is displayed.

First, we will change the link in the <c:action> in nonProtected.jsp. It should be linked to protected.jsp instead of loginForm.jsp.

Listing 9. Action in Non-protected JSP that directly links to protected.jsp.

```

<c:action name="Access protected page" link="protected.jsp">
    This use case starts when the user accesses a protected page.
    The system displays a login form.
</c:action>
  
```

Since the user doesn't log in yet at this moment, when the user accesses the protected.jsp, the page refuses to display the content. With J-CASE, the logic can be defined as a Java method in a JSP declaration tag (<%! %>).

Listing 10. protected.jsp with validate method, screens and actions tags.

```

<%!
@JCaseProcess("The system validates the session.")
public String validate(PageContext pageContext) throws Exception {
    String session = JCaseUtils.getParameter(pageContext, "session");

    if (!"valid".equals(session)) {
        throw new Exception("The session doesn't exist or timed-out.");
    }

    JCaseUtils.setAttribute(pageContext, "validationResult", "success");
    return "The user has already logged in.";
}
%>

<c:processResults /><br>

<c:choose>
    <c:when test="${validationResult == 'success'}">
        <c:screen name="Protected Page">
            <c:inst>Display contents the user requested.</c:inst>
        </c:screen>
    </c:choose>
  
```

```

        <c:actions>
            <c:action name="Display the protected page">
                The system validates the session,
                and displays the protected page requested.
                The use case ends.
            </c:action>
        </c:actions>
    </c:when>
    <c:otherwise>
        <c:screen />
        <br>
        <c:actions>
            <c:action name="Session doesn't exist or Timed-out"
                link="loginForm.jsp?target=protected.jsp">
                The session doesn't exist or timed-out.
                The system displays login page.
            </c:action>
        </c:actions>
    </c:otherwise>
</c:choose>

```

The method annotated with @JCaseProcess is automatically executed and we can see the execution result using <c:processResults> tag. In case of no session, the tag generates the process results table shown in Figure 8.

Figure 8. Process Results table that shows execution result of validate method.

Process Results (Success:0 Error:1) [hide]			
No.	Name	Result	Message
1	validate	Error	The system validates session. The session doesn't exist or timed-out.

In this case, the method throws an exception before setting "success" into validationResult attribute in PageContext, so the contents of <c:screen> and <c:action> tags in <c:otherwise> are displayed. Also the target parameter in the link attribute specifies the URL that has to be displayed when the authentication is complete successfully.

The loginForm.jsp is same as the JSP in previous storyboard but adding target parameter passed by the protected.jsp

Listing 11. loginForm.jsp that target parameter is added.

```

<c:action name="Correct Credencial"
    link="loginProcess.jsp?id=correctId&password=correctPwd&target=${param.target}">
    The user enters ID and password.
</c:action>

```

The next JSP, loginProcess.jsp, executes authenticate method to check to see if the ID and password are correct.

Listing 12. loginProcess.jsp that authenticates the user.

```

<%!
@JCaseProcess("The system authenticate the user.")
public String authenticate(PageContext pageContext) throws Exception {
    String id = JCaseUtils.getParameter(pageContext, "id");
    String password = JCaseUtils.getParameter(pageContext, "password");

```

```
        if (!"correctId".equals(id) || !"correctPwd".equals(password)) {
            throw new Exception("Invalid ID or password.");
        }

        JCaseUtils.setAttribute(pageContext, "authResult", "valid");
        return "Valid session has been created.";
    }
%>

<c:processResults /><br>

<c:actions test="{authResult == 'valid'}">
    <c:action name="Display the target page"
        link="{param.target}?session=valid">
        The system authenticates the user,
        and navigates the user to the target page with a valid session.
    </c:action>
</c:actions>

<c:actions test="{authResult != 'valid'}">
    <c:action name="Display login page again"
        link="loginForm.jsp?target={param.target}">
        The system failed to authenticate the user,
        and displays an error message and a link to login page again.
    </c:action>
</c:actions>
```

In case of correct ID and password, the authenticate method populates "valid" into authResult attribute. Then it shows the action linked to the target URL with "session=valid", which represents the session has been created successfully. This time, the protected.jsp accepts the request to show the protected contents because of the valid session.

Since we used a simple use case here, it looks easy to trace the flows even if we describe on paper or presentation tools. But, in actual software design, we usually have to also consider another use cases that are tightly related. For example, in case of the "Log in" use case, we need to consider "Forgot password" and "Create a new account" cases as well. J-CASE makes it easy to trace such a complex flow by executing JSPs and adding screens and process results in the documents.

4 Process Methods

In Use Case, we don't usually specify system processes, but they are specified in Storyboards. With J-CASE, the processes to be executed on server site and client side can be express using regular Java method.

If you want to describe a process to validate user ID and password, you can define Java methods in a declaration tag (<%! %>) of your JSP. Also in order to tell J-CASE that the method is the process to be executed, @JCaseProcess annotation with a description needs to be added or the method name should start with 'jcase_' (The prefix can be changed using processPrefix attribute in jcase.xml).

The format of the method is as follows:

```
@JCaseProcess("Description")
public void methodName(PageContext pageContext) throws Exception;
```

or

```
public void jcase_methodName(PageContext pageContext) throws Exception;
```

If the process is executed successfully, the following row is displayed in process results frame: (Note that description is not displayed if you use jcase_ prefix to define it as JCaseProcess)

Process Results (Success:1 Error:0)				
No	Name	Result	Description	Message
1	methodName	Success	Description	

If you want show a result message, you can use the format to return the message. (The type of return value is "String")

```
@JCaseProcess("Description")
public String methodName(PageContext pageContext) throws Exception {
    ....
    return "Logged in successfully."
}
```

If the process is executed successfully, the return value is displayed in Message field.

Process Results (Success:1 Error:0)				
No	Name	Result	Description	Message
1	methodName	Success	Description	Logged in successfully

If you throw an exception with a message in the method, the execution of the process is failed and the message is displayed in Message field in process results frame.

```
throw new Exception("Failed to log in.");
```

Process Results (Success:0 Error:1)				
No	Name	Result	Description	Message
1	methodName	Error	Description	Failed to log in.

The below shows sample methods that represent validation for ID and password:

```
<%@ page import="jcase.*" %>
```

```

<%!
@JCaseProcess("Validate on client-side")
public void p01_validateOnClient(PageContext pageContext) throws Exception {
    String email = JCaseUtils.getParameter(pageContext, "email");
    String password = JCaseUtils.getParameter(pageContext, "password");

    if (email == null || password == null) {
        JCaseUtils.setAttribute(pageContext, "loginResult", "errorOnClient");
        throw new Exception("Invalid format of email or password.");
    }
}

@JCaseProcess("Validate on server-side")
public void p02_validateOnServer(PageContext pageContext) throws Exception {
    String email = JCaseUtils.getParameter(pageContext, "email");
    String password = JCaseUtils.getParameter(pageContext, "password");

    if (!"test@test".equals(email) || !"testpass".equals(password)) {
        JCaseUtils.setAttribute(pageContext, "loginResult", "errorOnServer");
        throw new Exception("Failed to log in.");
    }

    JCaseUtils.setAttribute(pageContext, "loginResult", "success");
}
}
%>

```

The method must be public and has a parameter that is PageContext. In case of error, you can throw an Exception with a message.

If you declare multiple methods, each method should start with a prefix that represents an order to be executed. J-CASE doesn't execute the methods according to the order of the place they are declared in the JSP.

Once you declare the methods, J-CASE automatically executes, then you can see the process results using <c:processResults> tag. If the second method failed, you will see the following results:

Process Results (Success:1 Error:1)				
No	Name	Result	Description	Message
1	p01_validateOnClient	Success	Validate on client-side	
2	p02_validateOnServer	Error	Validate on server-side	Failed to log in.

5 Custom Style sheet

J-CASE includes the style sheet defined in `jcage/config/resource/jcage_system.css` and `jcage_default.css` in the generated HTML where you put `<c:head>` tag in Jsp page. If you want to add own style, you can create your style sheet with the name "jcage.css", and put it in your CLASSPATH. The steps are as follows:

1. Create `jcage.css` file.
2. Place it in a directory in CLASSPATH of your project.
(e.g. `<TOMCAT_HOME>/webapps/<Context-Name>/WEB-INF/classes/`)

6 User JavaScript

If you want to include your own JavaScript in all generated HTML, you can define your JavaScript in `jcage.js` file and place it in a directory in CLASSPATH of your project. It is outputted where you insert `<c:head>` tag in your JSP.

7 jcase.xml

This chapter describes how you can change configuration values of the system and each tag, and define own user attributes that can be referred through EL in JSP.

To configure your own values, follow the steps below:

1. Create jcase.xml file.
2. Place it in a directory in CLASSPATH of your project.
(e.g. <TOMCAT_HOME>/webapps/<Context-Name>/WEB-INF/classes/)

The default values of all attributes are defined in jcase/config/resource/jcase.xml in jcase.jar file.

7.1 System Attributes

The system attributes are attributes that the change affects to all tags. The attributes you can configure are as follows:

Name	Description	Default
frameBarBgcolor	Background color of frame bar.	#C0C0C0
frameBorderColor	Color of frame border.	empty
frameBorderColorDark	Dark color of frame border.	empty
frameBorderColorLight	Light color of frame border.	#C0C0C0
tableTitleBgcolor	Background color of table title show in frame content	lightgrey
tableBodyBgcolorDark	Dark background color of table body show in frame content.	#F7F7F7
tableBodyBgcolorLight	Light background color of table body show in frame content.	white
databaseFile	File path for the database that feedback comments are stored.	empty
processPrefix	Prefix of JCaseProcess method.	jcase_
showTagId	Pop a tooltip to display Tag ID true : Pop a tooltip false : No tooltip for Tag ID	false

7.2 Tag Attributes

Default values of attributes of most J-CASE tags can be changed. For example, <c:actions> tag are defined in system jcase.xml:

```
<tags>
  <tag name="actions">
    <attribute name="show">true</attribute>
    <attribute name="width">100%</attribute>
    <attribute name="border">1</attribute>
    <attribute name="title" />
  </tag>
</tags>
```

If you want change the width to '90%', you can specify it in your jcase.xml:

```
<tags>
  <tag name="actions">
    <attribute name="width">90%</attribute>
  </tag>
</tags>
```

The rest of attributes use the values defined in system jcase.xml.

7.3 User Variables

If you need to define own variables, which returns values you defined, in JSP, <user> tag can be defined in your jcase.xml.

```
<user>
  <attribute name="images">/mycase/images</attribute>
  <attribute name="myColor">pink</attribute>
</user>
```

The below shows the way to access the image and myColor attributes in JSP.

```
<c:image title="Insert CD">
  
</c:image>

<c:screen bgcolor="${user.myColor}">
  ...
</c:screen>
```

If you want to use another variable name instead of 'user', you can specify it in name attribute.

```
<user name="userVar">
  <attribute name="images">/mycase/images</attribute>
```

In this case, the defined value can be referred with "\${userVar.images}" in JSP.

If you doesn't want to specify the context path, '/mycase' in this sample, you can call getContextPath ServletAPI using EL.

```
<user>
  <attribute name="images">${pageContext.request.contextPath}/images</attribute>
</user>
```

Likewise, if you also need to specify absolute path including server name and port, getServerName and getServerPort API can be called.

```
<user>
  <attribute name="images">http://${pageContext.request.serverName}:
    ${pageContext.request.serverPort}
    ${pageContext.request.contextPath}/images</attribute>
</user>
```

The <user> tag can be described multiple times. For example, if you want to define lang variable that defines language colors, you can define <user name="lang"> as well as <user>.

J-CASE creates two variables, user and lang, and put them into the PageContext so that you refer with J-CASE tags.

```
<user>
  <attribute name="images">${pageContext.request.contextPath}/images</attribute>
</user>

<user name="lang">
  <attribute name="en">black</attribute>
  <attribute name="ja">pink</attribute>
</user>
```

Also, you can define child user variables in the <user> tag.

```
<user>
  <attribute name="images">${pageContext.request.contextPath}/images</attribute>
  <user name="screens">
    <attribute name="path">/mycase/screens.jsp</attribute>
    <attribute name="target">_blank</attribute>
  </user>
</user>
```

The path attribute in the nested <user> tag can be reached using the expression “`${user.screens.path}`” in JSP.

7.4 Use Case Names

Use Case names are usually referred in some places in JSPs, because <c:useCase> and <c:action> require to specify the use case name. When we developing JSPs, Use Case names are often changed. So, you can define use case names in jcase.xml. Once you define them using <useCase> tag, J-CASE tags retrieve use case names corresponding to the ID.

```
<useCases>
  <useCase id="UC01">Register User</useCase>
  <useCase id="UC02">Update Profile</useCase>
  <useCase id="UC03">Unsubscribe User</useCase>
</useCases>
```

Once you define the names in jcase.xml, you don't have to specify name attribute in <c:useCase> tag:

```
<c:useCase id="UC01" link="uc01/registerUser.jsp">
  ...
</c:useCase>
```

Also, you don't have to specify useCaseName attribute in <c:action> tag when you want to make a link to another use case:

```
<c:action name="Select Register" useCaseId="UC01" link="uc01/registerUser.jsp">
  ...
</c:action>
```

The useCases is also defined as user variable. So you can refer the use case name in JSP:

```
<c:inst>[UC01] #{useCases.UC01}</c:inst>
```

If you want to use another variable name, it can be specified in name attribute of useCases tag.

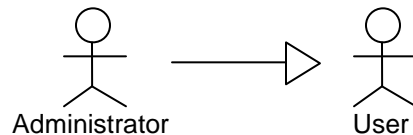
```
<useCases name="myUseCases">  
  <useCase id="UC01">Register User</useCase>  
  ...  
</useCases>
```

7.5 Actor Names

Actor names are also referred in some useCase tags repeatedly. So, you can define the actor names in jcase.xml. Once you define them using <actor> tag, J-CASE tags retrieve the names corresponding to the ID.

```
<actors>  
  <actor id="ACT01">User</actor>  
  <actor id="ACT02" generalizes="ACT01">Administrator</actor>  
</actors>
```

The second <actor> tag shows the usage of generalization between actors. It represents that the ACT02 (Administrator) is specialized based on the ACT01 (User).



Once you define the names in jcase.xml, actor ID can be specified in actor attribute in <c:useCase> tag:

```
<c:useCase id="UC01" actor="ACT01">  
  ...  
</c:useCase>
```

The actors are also defined as user variable. So you can also refer the actor name using EL in JSP:

```
<c:inst>Actor #{actors.ACT01}</c:inst>
```

8 Generate Use Case Document

Once you have created Use Case JSP and Action JSPs, you can create Use Case document from the JSPs. J-CASE provides two ways; `<c:generateUseCase>` tag and UseCaseGenerator command. (To use UseCaseGenerator command is highly recommended.)

Note that `<c:generateUseCase>` tag doesn't work properly on some servlet container, also it's not secure to use this tag on Web Server connected to Internet. Please use this tag only for internal use. We strongly recommend using the UseCaseGenerator Java application for other cases.

8.1 `<c:generateUseCase>` tag

J-CASE provides `<c:generateUseCase>` tag to generate a use case document in JSP. Required parameters are as follows:

- input
- output

So, if you create Use Case JSP that contains `<c:useCase>` tags as, for example, `uc/index.jsp`, the use case document can be generated with the two attributes.

```
<c:generateUseCase  
    input="uc/index.jsp"  
    output="useCase.html"  
/>
```

After executing the JSP that the above tag is described, you will see `useCase.html` is created in the same location.

However, in order to create the document with better style, the following attributes are effective:

- title
- width
- align
- cover
- pageBreakContents

The **title** is the title to be printed in header or footer of browser when printing the document.

The **width** is width of use case tables generated.

The **align** is the align of the tables. Left and center can be specified.

The **cover** is a HTML file to be included as a cover page of the document.

The **pageBreakContents** is a flag to insert a page break before contents page. If the number of use cases is not so many, it is better to specify it to false.

Also you may need to specify the following attributes in some cases:

- debug
- encoding
- user
- password

The **debug** is a flag to show debug messages. If the document is not generated properly, you can specify the debug to true. The messages tell you how many errors occurred and which pages were failed to access.

The **encoding** is the attribute to specify encoding, e.g. UTF-8.

The **user** and **password** are authentication information to access JSP pages. If you set up BASIC authentication for Use Case and Action JSPs, you must specify the user and password.

With the attributes, the `<c:generateUseCase>` looks like below:

```
<c:generateUseCase
  input="uc/index.jsp"
  output="useCase.html"
  title="Sample Use Cases"
  width="90%"
  align="center"
  cover="cover.html"
  pageBreakContents="false"
  debug="true"
  encoding="UTF-8"
  user="user"
  password="password"
/>
```

8.2 UseCaseGenerator command

J-CASE also provides UseCaseGenerator command to generate a use case document. It can be invoked from command line. `<c:generateuseCase>` tag is more convenient, because we don't have to upload document HTML generated. However, it has some concerns, for example, timed-out if taking longer time to generate the document, non-secure due to generating a file on server. So, formally we recommend using the UseCaseGenerator command instead.

The command also accept same parameters as `<c:generateUseCase>` tag. But there is two difference that are `urlHost` parameter is mandatory and you have to specify the URI starting with context name for input parameter.

```
Usage:
java jcase.tool.UseCaseGenerator
  -urlHost <Host and port>
  -input <URI-list>
  -output <file>
  [-cover <file-list>]
  [-appendix <file-list>]
  [-debug {true|false}] (default:false)
  [-title <Document title>]
  [-pageBreak {true|false}] (default:true)
  [-pageBreakContents {true|false}] (default:true)
  [-pageBreakAppendix {true|false}] (default:true)
  [-encoding <Encoding>]
  [-width <Width of tables>] (default:100%)
  [-align <Alignment of tables>] (default:left)
  [-debug {true|false}] (default:false)
  [-user <User name for Authorization>]
  [-password <Password of the user>]
  [-authHeader <HTTP Header value of Authorization>]
  [-showName {true|false}] (default:true)
  [-showScreen {true|false}] (default:true)
```

```
[-showProcessResults {true|false}] (default:true)  
[-showContents {true|false}] (default:true)  
[-useCaseDiagram <file>]  
[-navigationMap <file>]
```

The attributes we described in previous section for <c:generateUseCase> tag can be described for this command like below:

```
SET JAVA_HOME=\Progra~1\Java\jdk1.5.0_xx  
SET PATH=%JAVA_HOME%\bin  
SET CLASSPATH=<J-CASE HOME>/dist/jcase.jar  
  
java jcase.tool.UseCaseGenerator -urlHost http://localhost:8080  
-input /login-sample/uc/index.jsp -output useCase.html  
-title "Sample Use Cases" -width "90%" -align center -cover cover.html  
-pageBreakContents false -debug true -encoding UTF-8  
-user user -password password
```

NOTE: If you describe it in batch file of MS-DOS, % has to be described twice.

Also, J-CASE provides Ant task to execute this command from build script. The above command line can be represented below:

```
<?xml version="1.0"?>  
<project name="UseCaseGeneratorTask" default="generate">  
  
  <taskdef name="useCaseGenerator"  
    classname="jcase.tool.UseCaseGeneratorTask"  
    classpath="<J-CASE HOME>/dist/jcase.jar"/>  
  
  <target name="generate">  
    <useCaseGenerator  
      urlHost="http://localhost:8080"  
      input="/login-sample/uc/index.jsp"  
      output="useCase.html"  
      title="Sample Use Cases"  
      width="90%"  
      align="center"  
      cover="cover.html"  
      pageBreakContents="false"  
      debug="true"  
      encoding="UTF-8"  
      user="user"  
      password="password"/>  
    </target>  
  
</project>
```

8.3 UML Diagrams

<c:generateUseCase> tag and UseCaseGenerator command also generate XML data files for Use Case Diagram and Navigation Maps. And you can edit the generated XMLs using UMLet modeling tool and generate the image files. Here are the steps to generate diagrams and to save as image file:

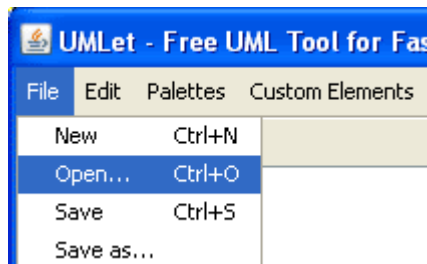
1. Generate UXF files (UML diagram files with UMLet format)
In <c:generateUseCase> tag, specify file path for the useCaseDiagram and navigationMap attributes. The extension must be ".uxf" so that UMLet can read the files.

```
<c:generateUseCase
  input="uc/index.jsp"
  output="useCase.html"
  useCaseDiagram="useCase.uxf"
  navigationMap="navMap.uxf"
/>
```

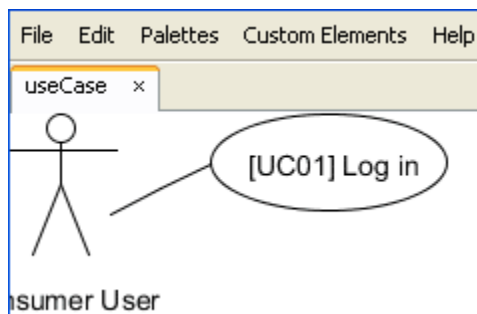
If you use UseCaseGenerator command, specify –useCaseDiagram and –navigationMap arguments with the same values.

```
java jcase.tool.UseCaseGenerator -urlHost http://localhost:8080
-input /login-sample/uc/index.jsp -output useCase.html
-useCaseDiagram useCase.uxf –navigationMap navMap.uxf
```

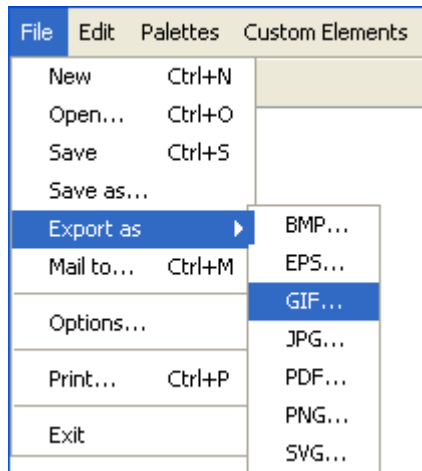
2. Load the UXF file
Run UMLet and select [File] – [Open] from the menu.
Select UXF file generated.



You will see the diagram generated by J-CASE.

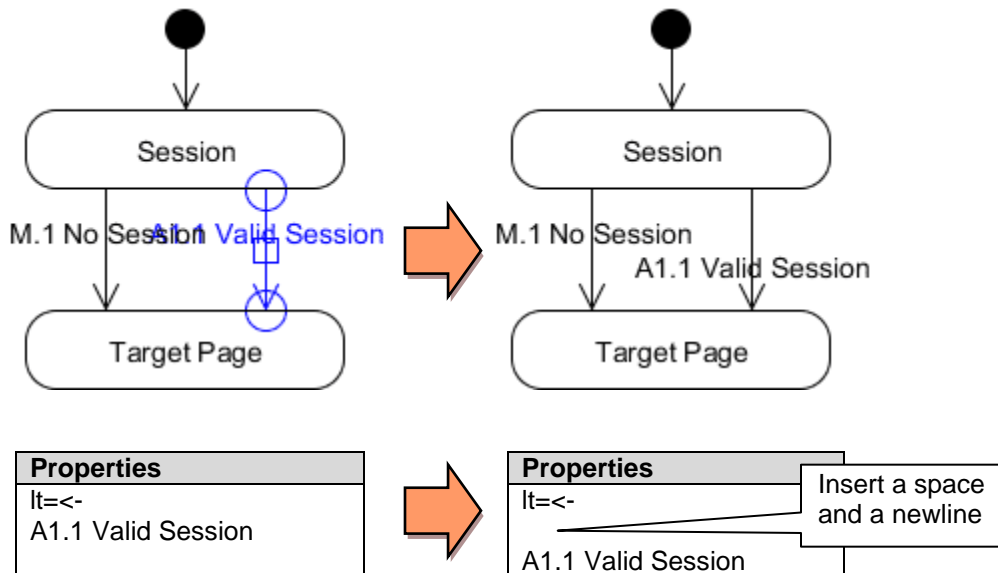


3. Edit the XML
Edit the UML diagram and save it.
4. Export as Image file
If you add the diagram into use case document, save it as an image file.
Select [File] – [Export as] from the menu and choose an image file format.



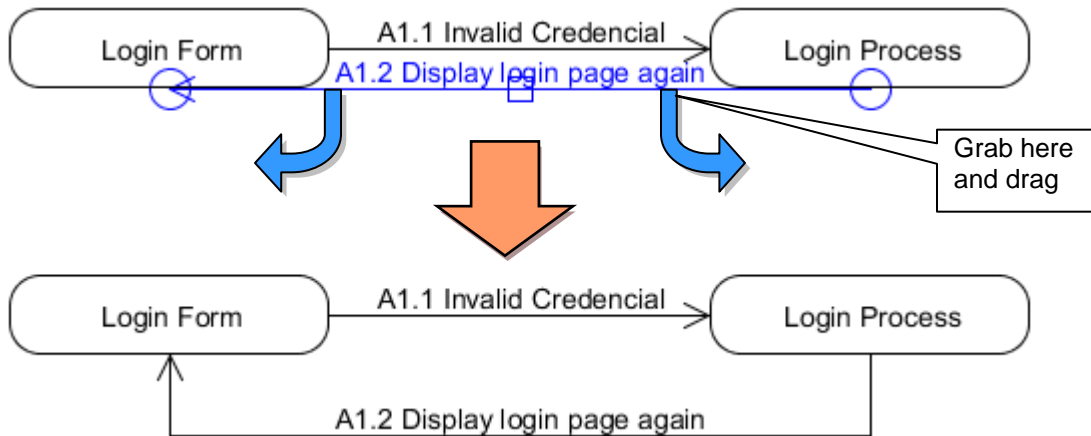
Tip 1: Label Location

To change the location of the label on a relation line, add spaces and/or new line in the properties. For example, if there are two associations and the labels are overwrapped, you can move the label down by adding a space and newline.



Tip 2: Make a Broken Line

To make a broken line, grab the line and stroke it where you want to make a break.



Tip 3: UML diagrams in Use Case document

The generated image files can be added in use case document using <c:chapter> and <c:section> tags. If you describe the following code in UseCase JSP, two chapters, "Use Case Diagram" and "Navigation Maps", which contains the UML diagram images, are added in Use Case Document:

```

<c:chapter name="Use Case Diagram">
  
</c:chapter>

<c:chapter name="Navigation Maps">
  <c:section name="[UC01] ${useCases.UC01}" /><br>
  <br>
  <br>
  <c:section name="[UC02] ${useCases.UC02}" pageBreak="true" /><br>
  <br>
  <br>
  <c:section name="[UC03] ${useCases.UC03}" /><br>
  <br>
  <br>
  <c:section name="[UC04] ${useCases.UC04}" pageBreak="true" /><br>
  <br>
</c:chapter>

```

9 Basic Authentication

If you don't want to public the use cases/storyboards for everyone, BASIC authentication can be used. This chapter shows how to configure the BASIC authentication for calculator sample running on Tomcat.

Two types of users are needed to define for the calculator sample (cal-sample.war), regular user and administrator. Although administrator can see all contents, the user cannot access generating use case document and feedback page with showing delete link.

The steps to set up are:

1. Add the following lines in <TOMCAT_HOME>/conf/tomcat-users.xml:

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="jcase_user"/>
  <role rolename="jcase_admin"/>
  <user username="user" password="user" roles="jcase_user"/>
  <user username="admin" password="admin" roles="jcase_user,jcase_admin"/>
</tomcat-users>
```

2. Confirm that the following lines are added in <TOMCAT_HOME>/webapps/cal-sample/WEB-INF/wex.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Admin Authentication</web-resource-name>
      <url-pattern>/admin/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>jcase_admin</role-name>
    </auth-constraint>
  </security-constraint>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>User Authentication</web-resource-name>
      <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>jcase_user</role-name>
      <role-name>jcase_admin</role-name>
    </auth-constraint>
  </security-constraint>
  <login-config>
    <auth-method>BASIC</auth-method>
```

```
<realm-name>Basic</realm-name>
</login-config>
<security-role>
  <role-name>jcase_user</role-name>
  <role-name>jcase_admin</role-name>
</security-role>

<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>

</web-app>
```

10 How to use JSTL

J-CASE only provides vary basic tags to control process flows, `<c:if>` and `<c:choose>`. In order to express more complex flows, JSTL (Java Standard Tag Libraries) can be used.

(NOTE: JSTL is not allowed to use in compiled JSPs for making a storyboard library.)

In order to enable to use the JSTL in your JSP, you first need to add JSTL JAR files (jstl.jar and standard.jar) in your lib directory.

Core tag library:

To use JSTL core tags, define uri and prefix for core tag using taglib directive:

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

<jstl:forEach var="item" items="{list}">
  ${item.key} = ${item.value}<br>
</jstl:forEach>
```

Functions:

To use JSTL functions, define uri and prefix for functions using taglib directive:

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

Escaped message: ${jstlfn:escapeXml(message)}<br>
Message length: ${jstlfn:toUpperCase(message)}<br>
```

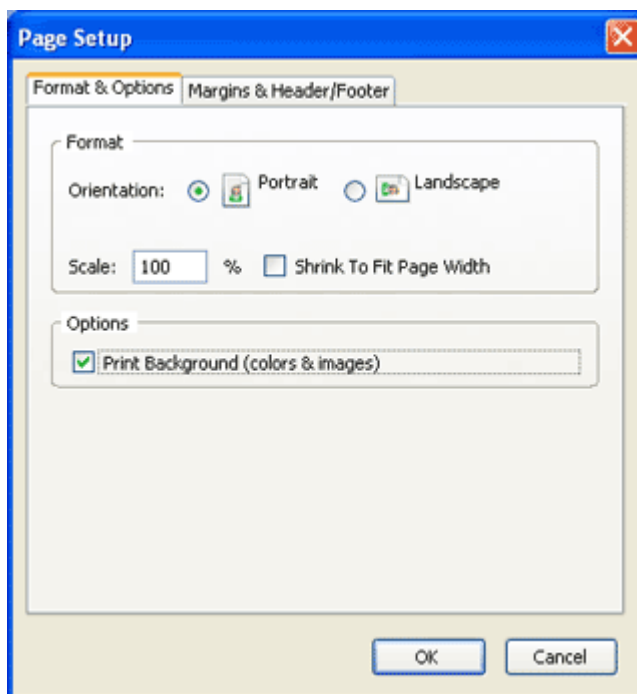
11 Tips

11.1 Print background colors and images

IE and Firefox with default settings don't print background colors and images. If you want to print use case documents generated by <c:generateUseCase> with background colors and images, follow the steps to change the setting:

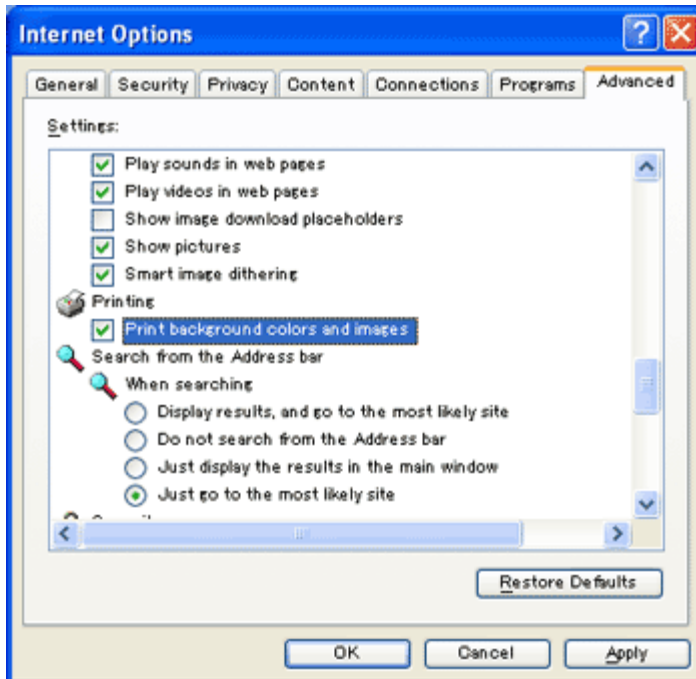
Firefox

1. Select File > Page Setup from the menu.
2. Page Setup dialog is opened. "Format & Options" tag is selected as a default.
3. Check "Print Background (colors & images)" in Options section
4. Click OK



Internet Explorer

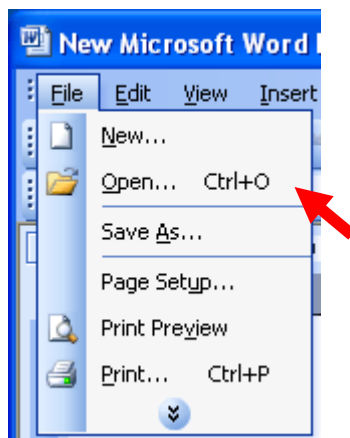
1. Select Tools > Internet Options from the menu.
2. Internet Options dialog is opened.
3. Select "Advanced" tag.
4. Check "Print background colors and images" in Printing section
5. Click OK



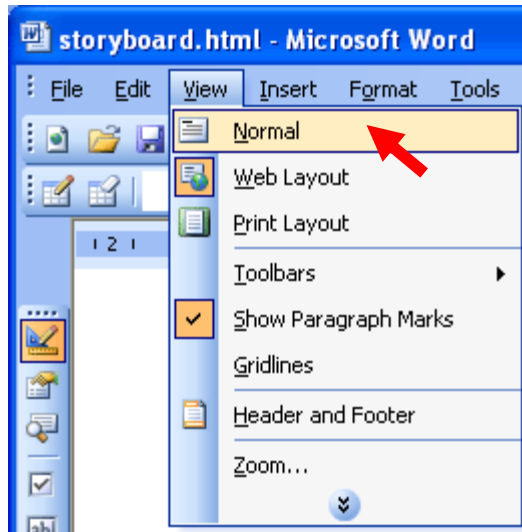
11.2 Convert the Use Case document into MS-WORD

The table of contents in the User Case document HTML file generated by <c:generateUseCase> or UseCaseGenerator command can be added page number. If you describe many use cases in the file, it's difficult to find the place on the printed document. In this case, you can convert it into Microsoft Word document. It allows you to easily add a table of contents with page numbers.

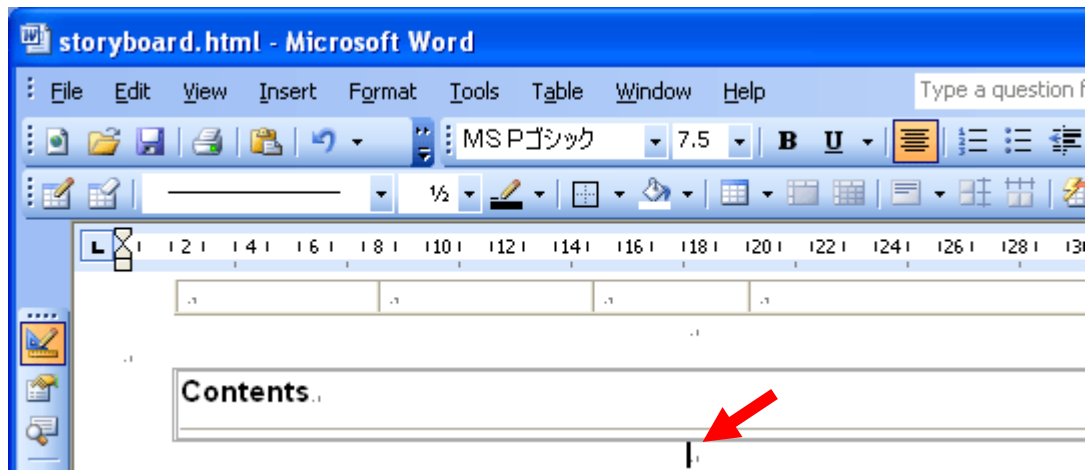
1. Generate a HTML of the Use Case document.
2. Open the HTML file generated from a Microsoft Word.



3. Change the view to "Normal" or "Print Layout".



4. Move your cursor where you want to insert a table of contents.



5. Select [Insert] → [Reference] → [Index and Tables] from the menu.
6. Select [Table of Contents] tab and click OK.
A table of contents will insert after the cursor.
7. Select [Insert] → [Reference] from the menu and click OK in the dialog.
Page number will be added in each page.

11.3 Authentication

In order to represent authentication mechanism in protected pages, authentication sample can be used. In the use case "UC01", login form is provided. Your JSPs to be protected simply need to include "authenticate.jsp".

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

If there is no session parameter with "valid" value, the JSP shows actions frame that navigates to the login form instead of showing your protected content.

11.4 URL Parameters to be ignored in generateUseCase

The <c:generateUseCase> tag checks to see if the link is already traced or not. If it is already traced, it stops tracing. However, the tag traces if there are any differences even if the link is logically same as a link already traced. In this case, "jcase_" URL parameters can be specified. URL parameters which name starts with "jcase_" are ignored in <c:generateUseCase>.

For example, there is a JSP which path is '/test.jsp'. You want to specify two actions in the JSP. One is a case that user enters a valid email. The other case is entering an invalid email. If you want to use a URL parameter to represent if an error happens, use parameter which name starts with "jcase_".

```
<c:if test="{param.jcase_err}">
  <font color="red">Please enter a valid email address.</font>
</c:if><br>
```

And the actions can be specified as follows:

```
<c:actions>
  <c:action name="Valid Email"
    link="Next.jsp?${fn:excludesQueryString(pageContext, 'jcase_err')}">
    The user enters a correct email.
  </c:action>

  <c:action name="Invalid Email"
    link="test.jsp?jcase_err=true&${fn:excludesQueryString(pageContext,
'jcase_err')}">
    The user enters an invalid email.
    The system displays an error message,
    e.g. "Please enter a valid email address."
  </c:action>
</c:actions>
```