Login Control example

1. overview

Login control is modification of login component. It is designed to be part of a larger *master page*. Unlike login dialog and login component, which can use any presentation technology, login control is based on JSP specification.

Converting a login component into a control is easy. We need to change the apperance of JSP pages, which have to blend with appearance of master page, and we need to make sure that the master page is reloaded when control changes its state.

2. Views

We need to redesign the appearance of JSP pages, and to make sure that they can fit the limited space on master web page. The easiest way is to design master page as table, and to put embedded control into table cell. Component is embedded into the master page using `<jsp:include>` directive. For example:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>

<html:html locale="true">
  ...
  <TABLE WIDTH=775 BORDER=0 CELLPADDING=0 CELLSPACING=0>
    ...
    <TR>
      <TD ROWSPAN=3>
        <IMG SRC="embedded-images/si1.gif" WIDTH=5 HEIGHT=104 ALT="">
      </TD>
      <TD COLSPAN=6 ROWSPAN=3 ALIGN="center" VALIGN="middle">

        <jsp:include page="/embeddedchild.do"/>

      </TD>
      <TD>
        <IMG SRC="embedded-images/spacer.gif" WIDTH=1 HEIGHT=12 ALT="">
      </TD>
    </TR>
    ...
  </TABLE>
</html:html>
```

As you can see, the login control is simply included in the master page, there no additional parameters, no special API to pass input data from master page to the control. The control simply renders itself when master page is reloaded.

3. Action Mapping

User input is submitted directly to login control instead of being channeled to the master page. How the login control can redraw itself, after its state changes?

The only change that we need to make to login control comparing to login component, is the location of DIALOG-RELOAD mapping. Now it will point to the master page. Notice, that this change is made in struts-config.xml file, so the component does not know that it renders within another page.

```
    <action path="/embeddedmasterpage" forward="/embedded-masterpage.jsp"/>
    <action path="/embeddedchild"
            type      = "net.jspcontrols.dialogs.samples.embedded.EmbeddedControl
            name      = "dialogloginform"
            scope     = "session"
            validate  = "false"
            parameter = "DIALOG-EVENT">
            <!-- Reloading master page -->
            <forward name = "DIALOG-RELOAD" path="/embeddedmasterpage.do" redirect="t
            <!-- Log In page -->
            <forward name="notloggedin" path="/embedded-child-login.jsp"/>
            <!-- Log Out page -->
            <forward name="loggedin" path="/embedded-child-logout.jsp"/>
    </action>
```

4. JSP compatibility

Is this that simple? Um, there is a small issue with JSP specification. JSP version 2.4, section 8.4 reads: "Before the forward method of the RequestDispatcher interface returns, the response content must be sent and committed, and closed by the servlet container." What does this mean? This means, that we cannot forward from login action class to JSP page. If we do, container has to close the response, and the rest of the master page below login control will not be rendered. Bummer. But there are solutions.

First of all, let us try, maybe it works. Tomcat 4.x and 5.x closes response as it supposed to according to JSP spec, which does not work for us. There is a patch, which keeps response opened. This patch can be used for development, but was not tested for production.

Another option is to use other containers. Resin works perfectly well. It does not close the response as spec mandates, but instead keeps it opened for included requests. This is what we need. But not everyone can use Resin.

Yet another, hackerish option is to use Java classes, generated from JSP pages. Jasper, which is included in Tomcat distribution, generates Java source code from JSP pages, and then compiles them. So, all we need is to copy generated Java files from Tomcat/work directory to the project directory, and to compile. In this case we will not perform server-side forwarding and Tomcat will be happy. This is how the online sample works, because my provides uses Tomcat.

Of course, hacking the JSP pages or container code is not the best choice, so I hope that JSP spec leads will hear these needs and will make a change in future JSP spec, which would say something like that: "Before the forward method of the RequestDispatcher interface returns, the response content must be sent and committed, and closed by the servlet container, IF REQUEST WAS NOT INCLUDED INTO ANOTHER REQUEST."

5. Live Demo

Login Control live demo