DialogAction example

1. Action class

DialogLoginAction handles input events, initializes dialog, and temporarily stores error messages between requests.

Login dialog can be initialized in a handler method. This means, that request should contain an initialization key, mapped to a method. You can use either a link or a submit button. For example, all following methods are valid for initialization:

```
<a href="dialogloginaction.do?DIALOG-EVENT-INIT=">Initialize login dialog</a>
<a href="dialogloginaction.do?DIALOG-EVENT-INIT=dummy">Initialize login dialog</a>
...
<form action="dialogloginaction.do" method="post">
  <input type="submit" name="DIALOG-EVENT-INIT" value="Initialize login dialog" />
</form>
```

Dialog is initialized from a page, external to dialog. For example, on the starting page you may have a link "Please, log in". It will initialize login dialog, and the following conversation will be handled by dialog itself. When login dialog is initialized, its `init` method is called. In this method dialog action clears username and password and logs out current user.

After dialog has initialized, it reloads itself, and renders a form with name and password fields. Because login dialog uses has only one view, default `getView` method is sufficient. By default, this method forwards to "DIALOG-VIEW" location, so all you need is to define the name of login JSP page.

When a user submits a form using "Log In" button, DIALOG-EVENT-LOGIN event is triggered and `login` method is called. It validates input, which is already stored in the corresponding form bean.

If login information is correct, login method redirects to user's home page. Otherwise `login` method generates error message, stores it in the session, and reloads dialog. The process repeats until correct user information is entered.

```
public class DialogLoginAction extends DialogAction {

   /**
    * Maps init event and submit buttons to handler methods
    */
   protected Map getKeyMethodMap() {
       Map map = new HashMap();
       // getInitKey() defined in SelectAction
       // to return "DIALOG-EVENT" prefix.
       map.put(getInitKey()-INIT, "init");
       map.put("DIALOG-EVENT-LOGIN", "login");
       return map;
   }

   /**
    * Initializes dialog
    */
   public ActionForward init (
           ActionMapping mapping,
           ActionForm form,
           HttpServletRequest request,
           HttpServletResponse response) throws Exception {

       DialogLoginForm inputForm = (DialogLoginForm) form;
       inputForm.setName(null);
       inputForm.setPassword(null);

       request.getSession().removeAttribute("login.username");

       return mapping.findForward(DialogConstants.DIALOG_RELOAD_KEY);
   }

   /**
    * Process login attempt (POST)
    */
   public ActionForward login (
           ActionMapping mapping,
           ActionForm form,
           HttpServletRequest request,
           HttpServletResponse response) throws Exception {

       // Validate user name and password
       ActionMessages errors = form.validate(mapping, request);

       // Errors found, stick them into session and reload the component
       if (errors != null) {
           saveDialogErrors(request.getSession(), errors);
           return mapping.findForward(DialogConstants.DIALOG_RELOAD_KEY);

       // Login OK, redirect to user's page
       } else {
           DialogLoginForm inputForm = (DialogLoginForm) form;
           request.getSession().setAttribute("login.username", inputForm.getName());
           return mapping.findForward("userhome");
       }
   }
}
```

2. Form Bean

Follows the corresponding form bean. Web components and dialogs use form bean as input/output buffer, therefore a form bean must have session scope to retain values between requests. Automatic validation must be turned off, so handler methods would be called even if input data is incorrect.

```
/**
 * Simple login form
 */
public class DialogLoginForm extends ActionForm {

   private String name;
   public String getName() {return name;}
   public void setName(String name) {this.name = name;}

   private String password;
   public String getPassword() {return password;}
   public void setPassword(String password) {this.password = password;}

   // Generate the error messages in the same manner as usual,
   // but do not forget to turn "validate" property of the action mapping off.
   public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
       if (!"guest".equalsIgnoreCase(name) ||
           !"guest".equalsIgnoreCase(password)) {
           ActionErrors errors = new ActionErrors();
           errors.add("ERROR", new ActionMessage("login.badpassword"));
           return errors;
       } else {
           return null;
       }
   }
}
```

3. Action Mapping

Finally the action mapping. Pay attention that "validate" is turned off, and that form bean has "session" scope. Also note two mandatory mappings: for component reloading and for the view.

Note that "input" property is not used, because action class has to be called despite of the result of input data validation. Validation is performed in the action class.

```
 <action path="/dialogloginaction"
         type      = "org.superinterface.samples.struts.dialogloginaction.DialogLoginA
         name      = "dialogloginform"
         scope     = "session"
         validate  = "false"
         parameter = "DIALOG-EVENT">
         <!-- Reload login form, if username/password combination is incorrect. -->
         <forward name = "DIALOG-RELOAD" path="/dialogloginaction.do" redirect="true"/
         <!-- The view corresponding to dialog. Recall, that dialogs have only one vie
```

```
          <forward name="DIALOG-VIEW" path="/dialogloginaction.jsp"/>
          <!-- Jump to user's home page -->
          <forward name="userhome" path="/dialogloginuserpage.jsp" redirect="true"/>
  </action>
```

4. Page Caching

Stale pages are not tolerated by Struts Dialogs library. At any time a web page must reflect current state of the application. This can be achieved by marking pages as non-cachable. Internet Explorer is satisfied with "no-cache" cache control header, while Mozilla/Firefox needs a stronger "no-store" header.

The easiest way to apply non-cachable headers to all pages of your application is to set `nocache` attribute to "true" in the controller element of `struts-config.xml`.

```
<controller nocache="true"/>
```

Struts started to use "no-store" header from version 1.2.4, which is the recommended version to use with Struts Dialogs library. Struts 1.2.2 is supported too, it this case use `setNoCache` method of DialogAction. Call this method in the first statement of your `getView` method.

5. Live Demo

Login Dialog live demo