Struts Dialogs Mail Reader: Home

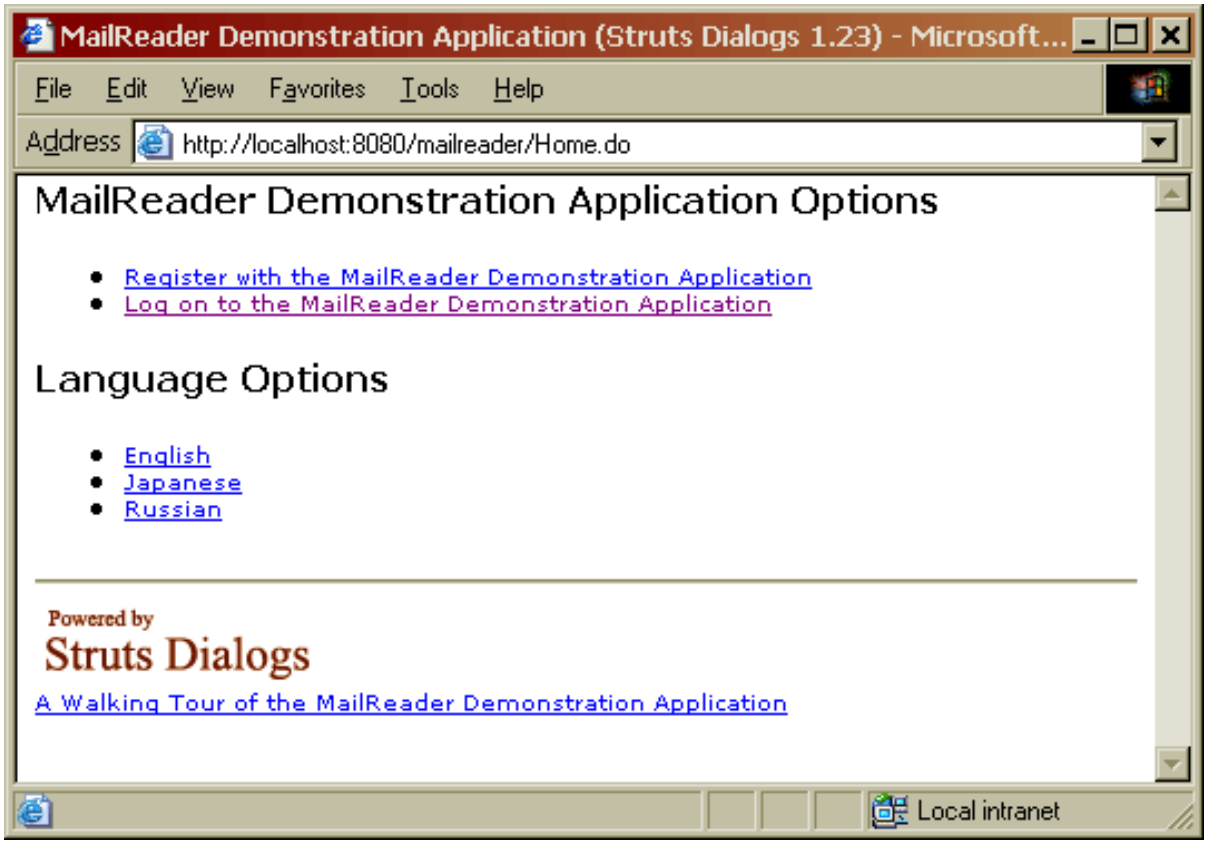
1. Overview

Home component carries out three functions:

- Startup menu
- Main menu
- Language selection

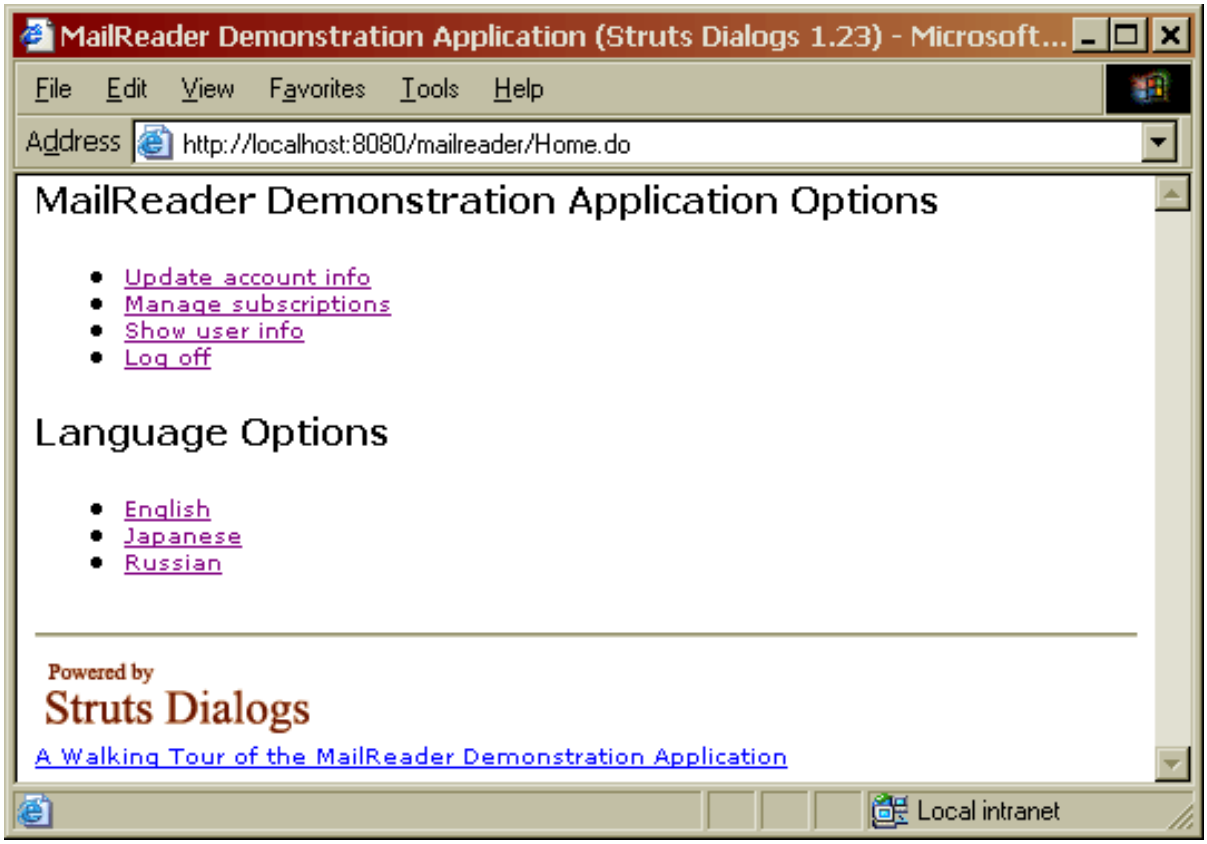
Home component is controlled by one action (HomeAction) and has one view (home.jsp). The `home.jsp` page displays either startup menu or main menu depending on user's logged-in status.

Startup menu is shown to a non-logged-in user and allows either to log in or to create a new user account.



MailReader Startup Menu

Main menu is shown to a logged-in user and allows to review and update user account information, to manage email subscriptions and to log off the application.



MailReader Main Menu

2. home.jsp

Let us first design the `home.jsp` page, then we will define action class and wire events to event handles. The `home.jsp` page displays different menu options depending on user's status.

```
<%@ 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" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>

<html>
  <head>
    <title><bean:message key="index.title"/></title>
    <link rel="stylesheet" type="text/css" href="base.css" />
  </head>
  <body>
```

```
<h3><bean:message key="index.heading"/></h3>

<!-- Signup and login options for not logged-in user -->
<logic:notPresent name="user" scope="session">
  <ul>
    <li>
      <html:link action="/Home?DIALOG-EVENT-SIGNUP">
        <bean:message key="index.registration"/></html:link>
      </li>
    <li>
      <html:link action="/Home?DIALOG-EVENT-LOGON">
        <bean:message key="index.logon"/>
      </html:link>
    </li>
  </ul>
</logic:notPresent>

<!-- Account modification and subscription list for logged-in user -->
<logic:present name="user" scope="session">
  <ul>
    <li>
      <html:link action="/Home?DIALOG-EVENT-ACCUPDATE">
        <bean:message key="index.updateaccount"/>
      </html:link>
    </li>
    <li>
      <html:link action="/Home?DIALOG-EVENT-SUBSCRIPTIONS">
        <bean:message key="index.managesubscriptions"/>
      </html:link>
    </li>
    <li>
      <html:link action="/Logon">
        <bean:message key="index.showuserinfo"/>
      </html:link>
    </li>
    <li>
      <html:link action="/Logon?DIALOG-EVENT-LOGOFF">
        <bean:message key="index.logoff"/>
      </html:link>
    </li>
  </ul>
</logic:present>

<!-- Language selection -->
<h3><bean:message key="index.langopts"/></h3>

<ul>
  <li>
    <html:link action="/Home?DIALOG-EVENT-LOCALE&language=en">
      <bean:message key="index.langenglish"/>
    </html:link>
  </li>
  <li>
    <html:link action="/Home?DIALOG-EVENT-LOCALE&language=ru" useLocalEncoding="true"
```

```
        <bean:message key="index.langrussian"/>
      </html:link>
    </li>
  </ul>

  <hr/>
  <p><html:img bundle="alternate" pageKey="strutsdialogs.logo.path" altKey="strutsdialo
</body>
</html>
```

The `home.jsp` page has three sections. The language selection section is always displayed. The startup menu is displayed when a user is not logged in; it is wrapped into `<logic:notPresent name="user" scope="session">` element. The main menu is displayed to logged in user; it is wrapped into `<logic:present name="user" scope="session">` element. This works because when a user logs in, the "user" bean with user's account information is saved in the session.

3. Login state and events

The links in each section of `home.jsp` page generate events. An event is a HTTP request identified with dialog event parameter. By default, dialog event parameter name should start with "DIALOG-EVENT", as seen in the code above. Of course, it is possible to pass additional request parameters along, for example to change interface language to Russian: `/Home?DIALOG-EVENT-LOCALE&language=ru`

The recommended practice is to send all events to a parent action, in case of `home.jsp` page this would be HomeAction class. For example, `/Home?DIALOG-EVENT-LOGON` generates logon event, while `/Home?DIALOG-EVENT-SIGNUP` generates signup event. The benefit of sending all page events to a parent action is centralized processing and cleaner application structure.

On the other hand, nothing prevents from sending an event to a different action class, like you see in the case of logging user off: `/Logon?DIALOG-EVENT-LOGOFF`. Here the logoff event is sent directly to Logon action.

4. Home action

To properly handle input events and output page rendering, the home action must subclass a DialogAction class. Then request events are mapped to method handlers, `getInitKey` returns "DIALOG-EVENT" by default:

```
public final class  HomeAction extends DialogAction {
  ...
  protected Map getKeyMethodMap() {
    Map methodMap = new HashMap();
    methodMap.put(getInitKey()+"-LOGON", "onLogon");
```

```
    methodMap.put(getInitKey()+"-SIGNUP", "onAccountSignup");
    methodMap.put(getInitKey()+"-ACCUPDATE", "onAccountUpdate");
    methodMap.put(getInitKey()+"-LOCALE", "onLocale");
    methodMap.put(getInitKey()+"-SUBSCRIPTIONS", "onSubscriptions");
    return methodMap;
  }
  ...

}
```

Then event handlers are implemented. They must have the same signature as `Action.execute` method. For example, this is how account update handler is coded:

```
public ActionForward onAccountUpdate(ActionMapping mapping,
                                     ActionForm form,
                                     HttpServletRequest request,
                                     HttpServletResponse response)
   throws Exception {
   return mapping.findForward("accupdate");
}
```

Method `onAccountUpdate` does not do much. It simply forwards the request to another resource called "accupdate".

The hanlder of language-change event is more elaborate. It obtains the desired language either from request parameter or from request header, sets the "locale" key in the session and redisplays `home.jsp` page:

```
public ActionForward onLocale(ActionMapping mapping,
                              ActionForm form,
                              HttpServletRequest request,
                              HttpServletResponse response)
    throws Exception {

  String language = request.getParameter("language");
  String country = request.getParameter("country");

    Locale locale = getLocale(request);
    if ((!isBlank(language)) && (!isBlank(country))) {
      locale = new Locale(language, country);
    } else if (!isBlank(language)) {
      locale = new Locale(language, "");
    }

    // Set current locale
    HttpSession session = request.getSession();
    session.setAttribute(Globals.LOCALE_KEY, locale);

    // Redisplay home page
    return EventForward.DIALOG_RELOAD;
   }
```

See the description for DialogAction class for in-depth explanation of how Struts Dialogs

components process input events and generate output.

5. Configuration of Home component

Follows the configuration of Home component in `struts-config.xml` file. The example below uses new **component**, **transfer** and **render** elements, introduced in Struts Dialogs 1.24:

```
<action-mappings>
   ...
   <!-- Home page. Handles welcome menu, main menu and language selection -->
   <component path = "/Home"
              view = "/home.jsp"
              type = "net.jspcontrols.mailreader.HomeAction">
       <transfer name = "logon" path = "/Logon.do?DIALOG-EVENT-INIT"/>
       <transfer name = "signup" path = "/Registration.do?DIALOG-EVENT-CREATE"/>
       <transfer name = "accupdate" path = "/Registration.do?DIALOG-EVENT-UPDATE"/>
       <transfer name = "subscriptions" path = "/Subscriptions.do?DIALOG-EVENT-INIT"/>
       <render name = "failure" path = "/error.jsp" />
   </component>
   ...
</action-mappings>
```

The same component can be configured using standard Struts **action** and **forward** elements, though with less clarity:

```
<action-mappings>
   ...
  <action path = "/Home"
          scope = "session"
          validate = "false"
          type = "net.jspcontrols.mailreader.HomeAction">
    <forward name = "logon" path = "/Logon.do?DIALOG-EVENT-INIT" redirect = "true"/>
    <forward name = "signup" path = "/Registration.do?DIALOG-EVENT-CREATE" redirect =
    <forward name = "accupdate" path = "/Registration.do?DIALOG-EVENT-UPDATE" redirect
    <forward name = "subscriptions" path = "/Subscriptions.do?DIALOG-EVENT-INIT" redir
    <forward name = "DIALOG-VIEW" path = "/home.jsp"/>
    <forward name = "failure" path = "/error.jsp"/>
  </action>
   ...
</action-mappings>
```

HomeAction has only one corresponding view, `home.jsp`, therefore HomeAction takes advantage of "view" attribute of "component" element, which defines the view for a component. If traditional syntax for action configuration is used, the view should be defined as result of a forward element with "DIALOG-VIEW" name. This is one of the default view mapping names, that is used by DialogAction.

Similarly, `<transfer>` element is just the same as `<forward ... redirect = "true">`, while `<render>` element is the same as `<forward ... redirect =`

"false">. Also, <component> element has different defaults, like turned off validation and session scope for form beans.

To use new syntax in `struts-config.xml`, you need to use Struts Dialogs 1.23 and to add new RuleSet object to `web.xml` file:

```
<init-param>
  <param-name>rulesets</param-name>
  <param-value>net.jspcontrols.dialogs.actions.DialogRuleSet</param-value>
</init-param>
```

If you have config validation turned on, then you need to have a proper DTD file `struts-config_1_2_dialog.dtd` for updated `struts-config.xml` file. If you place DTD file in WEB-INF directory, then you can refer to it using the following doctype in the struts-config.xml file:

```
<!DOCTYPE struts-config SYSTEM "struts-config_1_2_dialog.dtd">
```

The MailReader application retains a list of users along with their email accounts. The application stores this information in a database. If the application can't connect to the database, the application can't do its job. So before displaying the home page, the class checks to see if the database is available. The MailReader is also an internationalized application. So, the WelcomeAction checks to see if the message resources are available too. If both resources are available, the class displays home page. Otherwise, it forwards to the "failure" path so that the appropriate error messages can be displayed:

```
public ActionForward execute(ActionMapping mapping,
                             ActionForm form,
                             HttpServletRequest request,
                             HttpServletResponse response)
  throws Exception {

  // Setup message array in case there are errors
  ArrayList messages = new ArrayList();

  // Confirm message resources loaded
  MessageResources resources = getResources(request);
  if (resources == null) {
    messages.add(Constants.ERROR_MESSAGES_NOT_LOADED);
  }

  // Confirm database loaded
  UserDatabase userDatabase = getUserDatabase(request);
  if (userDatabase == null) {
    messages.add(Constants.ERROR_DATABASE_NOT_LOADED);
  }

  // If there were errors, queue plain error messages to the request,
  // and forward to boot failure page. Notice: no redirection here.
```

```
  if (messages.size() > 0) {
    request.setAttribute(Constants.ERROR_KEY,messages);
    mapping.findForward(Constants.FAILURE);
  }

  // Messages and database initialized successfully, let DialogAction
  // handle input and output.
  return super.execute(mapping, form, request, response);
}
```

6. Next: Login component

Next: the Login component.