ReusableClayJars

ClayAndTiles shows how Clays supports virtual entry points to pages in the form of full XML views. Full XML views allow you to reference a clay component which in turn references a physical file. This indirection is a nice way to abstract resources. You can use a simple name like /foo.xml to represent something that is really living in a location on the classpath, classpath*:org/bar/baz/foo.html, or under the web context, /pages/bar/baz/foo.html.

Unfortunately, if you like using the Tapestry-like full HTML views then you will soon find that mixing full HTML and full XML entry points can be troublesome. This example shows how one could package a Login Module as a self contained jar file using Clay full HTML views only. Then it shows how one could easily drop this jar into any JSF web application.

(1) First create a Java Project for the Login Module

(2) Create a package inside the project called loginmodule.pages

(3) Create a file called login.html in the loginmodule.pages package

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Login Page</title>
</head>
<body>
<span jsfid="messages"/>
<form jsfid="form">
        <h2>Login</h2>
        <table>
                <tr>
                        <td>Username:</td>
                        <td><input jsfid="lm:username" type="text"/></td>
                </tr>
                <tr>
                        <td>Password:</td>
                        <td><input jsfid="lm:secret" type="password"/></td>
                </tr>
                <tr>
                        <td><input jsfid="lm:loginButton" type="submit" value="Login"/></td>
                        <td><input jsfid="lm:cancelButton" type="submit" value="Cancel"/></td>
                </tr>

        </table>
</form>
</body>
</html>
```

(4) Create a file called login_success.html in the loginmodule.pages package

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Login Successful</title>
</head>
<body>
<form>

<h2>You have successfully logged in!</h2>

<p>Click <a jsfid="lm:continueLink" href="#">Here</a> to proceed</p>

</form>

</body>
</html>
```

(5) Create a directory named META-INF under the root of the project

(6) Add the following clay-config.xml to the META-INF directory

```
<view>

	<component jsfid="lm:username" extends="inputText">
		<attributes>
			<set name="value" value="#{userCredential.username}"/>
		</attributes>
	</component>

	<component jsfid="lm:secret" extends="inputSecret">
		<attributes>
			<set name="value" value="#{userCredential.secret}"/>
		</attributes>
	</component>

	<component jsfid="lm:loginButton" extends="commandButton">
		<attributes>
			<set name="value" value="Login"/>
			<set name="action" value="#{login.doLogin}"/>
		</attributes>
	</component>

	<component jsfid="lm:cancelButton" extends="commandButton">
		<attributes>
			<set name="value" value="Cancel"/>
			<set name="action" value="loginmodule.cancel"/>
		</attributes>
	</component>

	<component jsfid="lm:continueLink" extends="commandLink">
		<attributes>
			<set name="action" value="loginmodule.done"/>
		</attributes>
	</component>

</view>
```

(7) Next add the following faces-config.xml to the META-INF directory

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
  "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
  "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">

<faces-config>

        <managed-bean>
                <managed-bean-name>userCredential</managed-bean-name>
                <managed-bean-class>
                        loginmodule.UserCredential
                </managed-bean-class>
                <managed-bean-scope>request</managed-bean-scope>
        </managed-bean>

        <managed-bean>
                <managed-bean-name>login</managed-bean-name>
                <managed-bean-class>loginmodule.Login</managed-bean-class>
                <managed-bean-scope>session</managed-bean-scope>
        </managed-bean>

        <navigation-rule>
                <from-view-id>/classpath*:loginmodule/pages/login.html</from-view-id>
                <navigation-case>
                        <from-action>#{login.doLogin}</from-action>
                        <from-outcome>loginmodule.loggedin</from-outcome>
                        <to-view-id>/classpath*:loginmodule/pages/login_success.html</to-view-id>
                </navigation-case>
        </navigation-rule>

        <lifecycle>
                <phase-listener>loginmodule.SecurityPhaseListener</phase-listener>
        </lifecycle>

</faces-config>
```

(8) Create the following 3 Java classes inside the loginmodule package

(a) UserCredential.java

```
package loginmodule;

public class UserCredential {

        private String username;

        private String secret;

        public String getSecret() {
                return secret;
        }

        public void setSecret(String secret) {
                this.secret = secret;
        }

        public String getUsername() {
                return username;
        }

        public void setUsername(String username) {
                this.username = username;
        }

}
```

(b) Login.java

```
package loginmodule;

import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;
import javax.faces.el.ValueBinding;

public class Login {

        private boolean loggedIn = false;

        public String doLogin() {

                FacesContext ctx = FacesContext.getCurrentInstance();

                ValueBinding vb = ctx.getApplication().createValueBinding(
                                "#{userCredential}");

                UserCredential credential = (UserCredential) vb.getValue(ctx);

                if ("admin".equals(credential.getUsername())
                                && "nimda".equals(credential.getSecret())) {

                        setLoggedIn(true);

                        return "loginmodule.loggedin";
                }

                else {

                        setLoggedIn(false);

                        FacesMessage msg = new FacesMessage("Invalid Login",
                                        "You are not authorized");
                        ctx.addMessage(null, msg);

                        return null;
                }

        }

        public boolean isLoggedIn() {
                return loggedIn;
        }

        public void setLoggedIn(boolean loggedIn) {
                this.loggedIn = loggedIn;
        }

}
```

(c) SecurityPhaseListener.java

```
package loginmodule;

import javax.faces.context.FacesContext;
import javax.faces.el.ValueBinding;
import javax.faces.event.PhaseEvent;
import javax.faces.event.PhaseId;
import javax.faces.event.PhaseListener;

public class SecurityPhaseListener implements PhaseListener {

        public void afterPhase(PhaseEvent event) {

        }

        public void beforePhase(PhaseEvent event) {

                FacesContext ctx = event.getFacesContext();

                ValueBinding vb = ctx.getApplication().createValueBinding("#{login}");

                Login login = (Login) vb.getValue(ctx);

                if (!login.isLoggedIn()) {

                        ctx.getViewRoot().setViewId(
                                        "/classpath*:loginmodule/pages/login.html");

                        ctx.renderResponse();

                }

        }

        public PhaseId getPhaseId() {

                return PhaseId.RENDER_RESPONSE;

        }

}
```

(9) Build the project and then export the project as a jar called login.jar.

(10) Add login.jar to any JSF web project by including it in the WEB-INF/lib directory.

(11) Add this snippet to the faces-config file in the web project

```
        <navigation-rule>
                <from-view-id>/classpath*:loginmodule/pages/*</from-view-id>
                <navigation-case>
                        <from-outcome>loginmodule.done</from-outcome>
                        <to-view-id>/application.html</to-view-id>
                </navigation-case>
        </navigation-rule>
```

Substituting the path to the first page of the web application for /application.html above.

(12) Finally, make sure that the javax.faces.DEFAULT_SUFFIX is mapped to .html and the Faces Servlet is mapped to the *.html extension in web.xml

```
        <context-param>
                <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
                <param-value>.html</param-value>
        </context-param>


        <servlet-mapping>
                <servlet-name>Faces Servlet</servlet-name>
                <url-pattern>*.html</url-pattern>
        </servlet-mapping>
```

(13) This approach shows that you can configure Clay Full HTML views as entry points, evening going so far as having jarred html views referenced in viewIds. Admittedly, the viewId looks kind of strange but it works. This approach can be very useful for portal applications. For example, in the case where there are multiple portlets with some piece of overlapping functionality.

References

20060921 user@shale post