IM Login 1.0 Component Specification

1. Design

This component provides login support for client, manager and administrator roles in the IM application. The application will be deployed under the Struts framework, so the main classes of this component are implementations of Struts actions.

Since all the three actions required by the requirement need some common configurable properties such as logger, where to forward when login succeed or fail, etc, a base action is defined to provide such properties.

When processing client login, a database table “category” should be accessed to check whether the specified discussion category is chattable, so the ClientLoginAction uses db connection factory to access the database directly.

The security manager component is used to authenticate user, it is developed to use EJB so this component should configure the EJB environment for it, two configurable properties are provided to let the user of this component to provide the information needed by EJB.

No custom exception defined in this component, the assembler can use inner tag <exception> with struts configuration to show specified pages when error occurs.

Logging is supported by using component logging wrapper, each login operation should log messages comprised user ID, time stamp, action taken and entity IDs effected.

As required by the Struts framework, the action classes in this component should be programmed in a thread safe manner. Especially note when initialize the db connection factory in ClientLoginAction, it is synchronized and should be called each time the action is executed.

1.1 Design Patterns

None used.

1.2 Industry Standards

JSP/Servlet

Struts 1.3.5

EJB

1.3 Required Algorithms

1.3.1 Unregistered client login:

1) Retrieve the parameters: category, type

2) Check whether the specified category is chattable:

```
if (!checkChattable(category)) {
      logMessage(...);
      return mapping.findForward(unchattableForward)
}
```

3) Create a properties map, using the parameters loaded from request:

```
Map properties = new HashMap();
properties.put(getCategoryKey(), request.getParameter("cat"));
```

```
properties.put(getFamilyNameKey(), request.getParameter("fname"));
// ...
```

4) Set the role property to "client":

```
properties.put(getRoleKey(), "client");
```

5) Create a new chat user profile:

```
ChatUserProfile profile = new ChatUserProfile(properties);
```

6) Set the type of the profile to "Unregistered":

```
// the user name is composed simply to fname + " " + lname, but
actually its value will be ignored.
ProfileKey key = new ProfileKey(fname + " " + lname, "Unregis-
tered");
profile.setProfileKey(key);
```

7) Save the profile in session:

```
request.getSession().setAttribute(getuserProfileKey(), profile);
```

8) log the action

9) `return mapping.findForward(getLoginSucceedForward());`

1.3.2 Registered client login:

1) Retrieve the parameters from the request:

```
String user = request.getPrameter("user");
String password = request.getParameter("pwd");
String category = request.getPrameter("cat");
```

2) Check whether the specified category is chattable:

```
if (!checkChattable(category)) {
      logMessage(...);
      return mapping.findForward(unchattableForward)
}
```

3) Authenticate the user:

```
TCSubject sub = getLoginRemote().login(user, password);
```

4) If an GeneralSecurityException is thrown in last step, log a failure login and return false;

5) Retrieve the chat user profile:

```
ChatUserProfile profile = getChatUserProfileMan-
ager().getProfile(user, "Registered");
```

6) Save the selected discussion category and client role in the profile.

```
profile.setProperty(getCategoryKey(), category);
profile.setProperty(getRoleKey(), "client");
```

7) Store the user profile in session:

```
request.getSession().setAttribute(getUserProfileKey(), profile);
```

8) log the action

9) `return mapping.findForward(getLoginSucceedForward());`

1.3.3 Manager login:

1) Retrieve the parameters from the request:

```
String user = request.getPrameter("user");
String password = request.getParameter("pwd");
```

2) Authenticate the user:

```
TCSubject sub = getLoginRemote().login(user, password);
```

3) If an GeneralSecurityException is thrown in last step, log a failure login and

```
return mapping.findForward(getLoginFailForward());
```

4) Using authorization manager to check permission:

```
getAuthorizationManager().authorize(user, principal, context);
// if error occurred then log a failure login and return map-
ping.findForward(getLoginFailForward());
```

5) Retrieve the chat user profile:

```
ChatUserProfile profile = getChatUserProfileMan-
ager().getProfile(user, "Registered");
```

6) Save the selected discussion category and client role in the profile.

```
profile.setProperty(getCategoryKey(), category);
profile.setProperty(getRoleKey(), "manager");
```

7) Store the user profile in session:

```
request.getSession().setAttribute(getUserProfileKey(), profile);
```

8) log the action

9) `return mapping.findForward(getLoginSucceedForward());`

1.3.4 Admin login:

1) Retrieve the parameters from the request:

```
String user = request.getPrameter("user");
String password = request.getParameter("pwd");
```

2) Authenticate the user:

```
TCSubject sub = getLoginRemote().login(user, password);
```

3) If an GeneralSecurityException is thrown in last step, log a failure login and

```
return mapping.findForward(getLoginFailForward());
```

4) Using authorization manager to check permission:

```
getAuthorizationManager().authorize(user, principal, context);
```

5) Store the authorization principal in session:

```
request.getSession().setAttribute(getUserProfileKey(), principal);
```

6) log the action

7) `return mapping.findForward(getLoginSucceedForward());`

1.4 Component Class Overview

LoginAction (abstract):
This is the base action of this component. It defines the common configurable properties of all the actions in this component.

The properties are almost initialized in ctor by being loaded from configuration manager, then never changed later. Sub action classes can use them to decide where to forward after login, under which key to store the user profile in the session, etc.

It also provide convenient methods to help sub classes to access chat user profile manager and EJB instances when using security manager.

A method for logging is provided to support logging for convenience.

This class is immutable and thread safe.

ClientLoginAction:
This class provides client login support. It is an extension of LoginAction, which defines the common properties needed when doing login.

As requirement, it provides several additional configurable properties such as the configuration of db connection factory and the key names needed when creating chat user profiles.

It needs to access a database table “category” directly to check whether the specified discussion category is chattable, so the db connection factory is used to support db operations.

There are three possible forwards after the execution of this action, they are all specified by configurable properties:

1. unchattableForward (default: “unchattable”): if the specified discussion category is unchattable.

2. loginSucceedForward (default: “loginSucceed”): if the login succeeded

3. loginFailFoward (default: “loginFail”): if the login failed.

After successful login, a chat user profile object will be stored to the session under a configurable key specified by the field userProfileKey.

This class is immutable and thread safe.

ManagerLoginAction:

This class provides manager login support. It is an extension of LoginAction, which defines the common properties needed when doing login.

There are two possible forwards after the execution of this action, they are all specified by configurable properties:

1. loginSucceedForward (default: “loginSucceed”): if the login succeeded

2. loginFailFoward (default: “loginFail”): if the login failed.

After successful login, a chat user profile object will be stored to the session under a configurable key specified by the field userProfileKey.

This class is immutable and thread safe.

AdminLoginAction:
This class provides administrator login support. It is an extension of LoginAction, which defines the common properties needed when doing login.

There are two possible forwards after the execution of this action, they are all specified by configurable properties:

1. loginSucceedForward (default: “loginSucceed”): if the login succeeded

2. loginFailFoward (default: “loginFail”): if the login failed.

After successful login, an authorization principal object will be stored to the session under a configurable key specified by the field userProfileKey.

This class is immutable and thread safe.

1.5 Component Exception Definitions

No custom exception is defined in this component. All underlying exceptions are thrown directly, the assembler can use inner <exception> tag in struts configuration file to catch them.

The possible exceptions may be thrown by this component are:

IllegalArgumentException (from java.lang):
If any argument of method execute is null, this exception is thrown. In fact it can never happen since the arguments can not be null, this is guaranteed by struts framework.

SQLException (from java.sql):
The ClientLoginAction needs to access a database table directly, so this exception may be thrown due to errors when accessing database.

DBConnectionException (from com.topcoder.db.connectionfactory):
The ClientLoginAction needs to access a database table directly, this is done by using db connection factory component, this is an exception that may be thrown by that component.

ConfigurationException (from com.topcoder.db.connectionfactory):
The ClientLoginAction needs to access a database table directly, this is done by using db connection factory component, this is an exception that may be thrown by that component.

RemoteException (from java.rmi):
The security manager which is used to do authentication uses EJB, this exception may be thrown for error occurs when using EJB.

NamingException (from java.naming):
When finding EJB instances for security manager, this exception may be thrown.

ConfigurationException (from com.topcoder.security.authorization):
The authorization component is used to check permissions when manager or administrator logins, to use it a AuthorizationManager must be created, its constructor may thrown this exception for configuration errors.

IllegalReferenceException (from com.topcoder.util.objectfactory):

If cannot properly match specifications given for ChatUserProfileManager to each other, or the properties are malformed.

InvalidClassSpecificationException (from com.topcoder.util.objectfactory):

If createObject method fails, the specification is not valid and can't be used to create an object.

SpecificationConfigurationException (from com.topcoder.util.objectfactory):

If ObjectFactory creation fails during the construction of ChatUserProfileManager.

ChatUserProfilePersistenceException (from com.topcoder.chat.user.profile):

If any persistence related error occurs with the ChatUserProfilePersistence.

ProfileKeyManagerPersistenceException (from com.topcoder.chat.user.profile):

If any ProfileKey related persistence error occurs while communicating with the Chat Profile component.

ProfileNotFoundException (from com.topcoder.chat.user.profile):

If the chat user profile is not present in the persistence during the execute method of login actions.

UnrecognizedDataSourceTypeException (from com.topcoder.chat.user.profile):

If the type sources provided to the Chat Profile is not registered with the

ChatUserProfileManager.

1.6 Thread Safety

This component is thread safe as required by the Struts framework. All the fields are initialized once and never changed later, the db connections and EJB instances are created each time an operation executes, no shared variable exists.

One thing to note is initializing the db connection factory in ClientLoginAction, it is not done in ctor but the first time need to access the database (in method execute), so we need a mechanism to provide thread safety to this instance. The method initDBFactory initializes the db connection factory, this method constructs the factory if it has not been, it is synchronized and called it each time the ClientLoginAction is executed, in this way the db connection factory is thread safe.

2. Environment Requirements

2.1 Environment

- At minimum, Java1.4 is required for compilation and executing test cases.
- Java 1.4 or higher

2.2 TopCoder Software Components

- Base Exception 1.0 was used to provide a base exception for custom exceptions.
- Configuration Manager 2.1.5 was used to provide configuration management.
- Security Manager 1.1 was used to provide authenticate the user.
- Chat User Profile 2.0 was used to create or retrieve chat user profiles.
- Authorization 2.1.1 was used to perform check against roles and permissions.
- DB Connection Factory 1.0 was used to access the table “category”.
- Logging Wrapper 1.2 was used to support logging.
- JNDI Context Utility 1.0 was used to simplify the process of finding the Security Manager EJB
- Object Factory 2.0.1 was used to create the ChatUserProfileManager

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

- JUnit : 3.8.2 : http://www.junit.org
- Mock EJB 0.6.2 : http://mockejb.sourceforge.net/
- MockRunner 0.3.8 : http://mockrunner.sourceforge.net/

NOTE: The default location for 3rd party packages is ../lib relative to this component installation. Setting the ext_libdir property in topcoder_global.properties will overwrite this default location.

3. Installation and Configuration

3.1 Package Name

com.cronos.im

3.2 Configuration Parameters

Namespace: com.cronos.im.LoginAction

Parameter	Description	Values
userProfileKey (optional)	Represents the session key for an object that hold the user's profile, default value is “userProfile”	“userProfile”
loginSucceedForward (optional)	Where to forward when login succeed, default value is “loginSucceed”	“loginSucceed”

Parameter	Description	Values
loginFailForward (optional)	Where to forward when login fail, default value is “loginFail”	“loginFail”
categoryKey (optional)	The key corresponds to the property category when creating chat user profile, default value is “category”	“category”
roleKey (optional)	The key corresponds to the property role when creating chat user profile, default value is “role”	“role”
authProviderURL(optional)	The URL used to access EJB when using security manager, If not present or empty will be set to null	http://localhost:80
authInitialContextFactory (optional)	The initial context factory used to access EJB when using security manager, If not present or empty will be set to null	“org.jnp.interfaces.NamingContextFactory”
authorizationManager Namespace(optional)	The namespace to build an authorization manager, if missed then set to default value "com.cronos.im.AuthorizationManager" .	"com.cronos.im.AuthorizationManager"
ObjectFactoryNS (optional)	The namespace that is used to construct an ObjectFactory, if missed then set to default value " com.cronos.im.ObjectFactory "	"com.cronos.im.ObjectFactory"
chatUserProfileManager (optional)	The name used to create ChatUserProfileManager using ObjectFactory, if missed then set to default value "chatUserProfileManager"	"chatUserProfileManager"

Namespace: com.*cronos*.im.ClientLoginAction

Parameter	Description	Values

Parameter	Description	Values
DBConnectionFactoryNS(optional)	Represents the session key for an object that hold the user's profile, default value is “DBConnectionFactoryNS”	“userProfile”
connectionName (optional)	The connection name used to obtain a connectin from the db connection factory when accessing the category table, default value is “connectionName”	“connection”
unchattableForward (optional)	Where to forward when the category is unchattable, default value is “unchattable”	“unchattable”
familyNameKey (optional)	The key corresponds to the property family name when creating chat user profile, default value is “familyName”	“familyName”
lastNameKey (optional)	The key corresponds to the property last name when creating chat user profile, default value is “lastName”	“lastName”
companyKey (optional)	The key corresponds to the property company when creating chat user profile, default value is “company”	“company”
titleKey(optional)	The key corresponds to the property title when creating chat user profile, default value is “title”	“title”
emailKey (optional)	The key corresponds to the property email when creating chat user profile, default value is “email”	“email”

3.3 Dependencies Configuration

The db connection factory should be configured under a namespace specified by the property DBConnectionFactoryNS.

The security manager and authorization manager should be configured, see components’ specification for details.

The chat user profile manager should be configured, see component specification for details.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

1. Prepare the configuration file

2. Configure struts

3. Configure dependent components such as security manager, chat user profile and authorization

4.3 Demo

1. A sample configuration:

```
<CMConfig>
   <Config name="com.cronos.im.LoginAction">
      <Property name="userProfileKey">
         <Value>userProfile</Value>
      </Property>
      <Property name="loginSucceedForward">
         <Value>loginSucceed</Value>
      </Property>
      <Property name="loginFailForward">
         <Value>loginFail</Value>
      </Property>
      <Property name="categoryKey">
         <Value>category</Value>
      </Property>
      <Property name="roleKey">
         <Value>role</Value>
      </Property>
      <Property name="authProviderURL">
         <Value>http://locahost</Value>
      </Property>
      <Property name="authInitialContextFactory">
         <Value>org.mockejb.jndi.MockContextFactory</Value>
      </Property>
      <Property name="ObjectFactoryNS">
         <Value>of.ChatUserProfileManager</Value>
      </Property>
      <Property name="chatUserProfileManager">
         <Value>chatUserProfileManager</Value>
      </Property>
   </Config>

   <Config name="com.cronos.im.ClientLoginAction">
      <Property name="DBConnectionFactoryNS">

<Value>com.topcoder.db.connectionfactory.DBConnectionFactoryImpl</Value>
      </Property>
```

```
      <Property name="connectionName">
         <Value>informix_connect</Value>
      </Property>
      <Property name="unchattableForward">
         <Value>unchattable</Value>
      </Property>
      <Property name="familyNameKey">
         <Value>familyName</Value>
      </Property>
      <Property name="lastNameKey">
         <Value>lastName</Value>
      </Property>
      <Property name="companyKey">
         <Value>company</Value>
      </Property>
      <Property name="emailKey">
         <Value>email</Value>
      </Property>
      <Property name="titleKey">
         <Value>title</Value>
      </Property>
   </Config>
</CMConfig>
```

2. A sample struts configuration:

```
<struts-config>
   <!-- ============ Exception handling ================== -->
   <global-exceptions>
      <exception
         type="java.lang.Exception"
         key="error.input"
         path="errors.jsp"/>
   </global-exceptions>
   <form-beans>
      <!-- ==================== Configure a test form
==================== -->
      <form-bean name="testForm"
type="org.apache.struts.action.DynaActionForm"/>
   </form-beans>
   <action-mappings>
      <!-- ================== Configure the client login action
================== -->
      <action path="/client-login"
          type="com.cronos.im.ClientLoginAction"
          name="testForm">
            <forward name="unchattable" path="/unchattable.jsp"/>
            <forward name="loginSucceed" path="/success.jsp"/>
            <forward name="loginFail" path="/failure.jsp"/>
      </action>
      <!-- ================ Configure the manager login action
================ -->
      <action path="/manager-login"
```

```
            type="com.cronos.im.ManagerLoginAction"
            name="testForm">
              <forward name="loginSucceed" path="/success.jsp"/>
              <forward name="loginFail" path="/failure.jsp"/>
        </action>
        <!-- ============== Configure the administrator login action
================ -->
        <action path="/admin-login"
            type="com.cronos.im.AdminLoginAction"
            name="testForm">
              <forward name="loginSucceed" path="/success.jsp"/>
              <forward name="loginFail" path="/failure.jsp"/>
        </action>
    </action-mappings>
</struts-config>
```

5. Future Enhancements

Provide struts forms definition with required login information.