Basic Struts Actions 1.0 Component Specification

1. Design

The TopCoder registration process to become a member of TopCoder is going to be redesigned. The main goal is to allow for a minimal set of required info that the user needs to enter to sign up (handle, email, and password).

The existing TC web registration process will ask a new user to enter a lot of data when he/she tries to register to TC web-site. The registration process takes a lot of time, so many users can simply break the registration process and leave non-registered

Thus there is a need of an easy to use and user friendly web-application for supporting registration on the new users and management of registered user profiles at TC web-site.

The main goal of this application is to simplify registration of the new users on TC web-site by minimizing the count of mandatory data fields for new account registration, and to improve usability of user profile management for the registered users. Any additional profile information will be requested from the user by the system as it is needed. For example, if a user registers to compete in an assembly contest the site would prompt them for any required info that they have not yet entered.

This component is responsible for implementing an authorization interceptor; login, logout and password recovery struts actions

1.1 Design Patterns

1.1.1 Strategy

BaseAction uses AuditDAO implementations which are pluggable.
BasePasswordAction uses PasswordRecoveryDAO implementations which are pluggable.
BaseUserDAOAwareAction uses UserDAO implementations which are pluggable.
LoginAction uses LoginRemote implementations which are pluggable.
ResetPasswordAction uses PrincipalMgrRemote implementations which are pluggable.

In addition, the actions are also used strategically by the Struts framework.

1.1.2 DAO

AuditDAO, PasswordRecoveryDAO, and UserDAO are the DAO used by this component.

1.1.3 Intercepting Filter

ThrottleInterceptor and AuthorizationInterceptor are Intercepting Filters.

1.1.4 MVC

Actions can be treated like the Controller part of the MVC pattern.

1.1.5 IoC

The configuration is done through Spring injection and the Spring context is loaded by the job. Therefore the Inversion of Control (IoC) pattern is used.

1.2 Industry Standards

XML
JSP
HTML
EJB

1.3 Required Algorithms

Please refer to TCUML method doc for details not listed below.

1.3.1 Logging

The component will log activity and exceptions using the Logging Wrapper in struts actions and interceptors, and the Logging Wrapper should be configured to use Log4j.

It will log errors at Error level, and method entry/exit information at DEBUG level.

Specifically, logging will be performed as follows, if logging is turned on.

- Method entrance and exit will be logged with DEBUG level.
  - Entrance format: [Entering method {*className.methodName}*]
  - Exit format: [Exiting method {*className.methodName}]*. Only do this if there are no exceptions.
- Method request and response parameters will be logged with DEBUG level
  - Format for request parameters: [Input parameters[{request_parameter_name_1}:{ request_parameter_value_1}, {request_parameter_name_2}:{ request_parameter_value_2}, etc.)]]
  - Format for the response: [Output parameter {response_value}] . Only do this if there are no exceptions and the return value is not void.
  - If a request or response parameter is complex, use its toString() method. If that is not implemented, then print its value using the same kind of name:value notation as above. If a child parameter is also complex, repeat this process recursively.
- All exceptions will be logged at ERROR level, and automatically log inner exceptions as well.
  - Format: Simply log the text of exception: [Error in method {*className.methodName}*: Details {*error details*}]

In general, the order of the logging in a method should be as follows:

1. Method entry
2. Log method entry
3. Log method input parameters
4. If error occurs, log it and skip to step7
5. Log method exit
6. If not void, log method output value
7. Method exit

The toString() method of the entity can be used for the logging.

1.3.2 Error Handling

The action and interceptors throw various exceptions. These exceptions will be caught and the user will be redirected to some error page. This is out of scope. This component only needs to throw exception without worrying about redirecting the user to the error page.

1.3.3 Session Timeout

The session timeout should be set to 120 minutes in web.xml during the integration phase.

1.3.4 JSP

The developers should implement the following JSP according to the provided struts.xml:
error.jsp – this will be a simple JSP that displays all action errors and field errors.
authorizationError.jsp – this will be a simple JSP that tells the user that he is not authorized to perform the operation.
login.jsp – this is the page that provides the form for username, password, “remember me” that allows the user to login.
recoverPassword.jsp – this is the page that allows the user to input handle or email to recover password.
recoverPasswordEmailSent.jsp – this is the page that tells the user the password recovery email is sent.
resetPasswordConfirmation.jsp – this is the page that tells the user that the new password is generated.

1.3.5 Validation

XML validation is used for simple validations. The XML validation files and the message resource bundle are already provided.

1.3.6 @PostConstruct

The method that is marked with <<@PostConstruct>> in TCUML should be set as the value of the “init-method” attribute in the bean declaration of the Spring application context file. See the attached “applicationContext.xml” for example. These methods don’t need to be added with the javax.annotation.PostConstruct.

1.3.7 Interceptor Order

Note that ThrottleInterceptor must run before AuthorizationInterceptor.

1.4 Component Class Overview

com.topcoder.web.reg.actions.basic:

LoginAction:

This action handles user login. It logs the user in using EJB, ensures the user's status is active, checks if the user email is activated, uses BasicAuthentication to set remember-me cookies, and finally redirect the user to specific result depending on whether it's the first time the user logs in, and if the user tries to go somewhere before login.

Thread Safety:
This class is not thread-safe because it's mutable. However, dedicated instance of struts action will be created by the struts framework to process the user request, so the struts actions don’t need to be thread-safe.

BaseUserDAOAwareAction:

This is the base action of all actions that uses UserDAO. It simply has a userDAO for subclasses to use.

Thread Safety:
This class is not thread-safe because it's mutable. However, dedicated instance of struts action will be created by the struts framework to process the user request, so the struts actions don’t need to be thread-safe.

BaseAction:

This is the base action of all actions of this component. It extends ActionSupport and implement SessionAware and ServletRequestAware to get access to request and session. It provides the method to get a WebAuthentication from the http session. It has a method that performs auditing.

Thread Safety:
This class is not thread-safe because it's mutable. However, dedicated instance of struts action will be created by the struts framework to process the user request, so the struts actions don’t need to be thread-safe.

LogoutAction:

This action handles user logout. It simply logs the user out with the WebAuthentication from session, and invalidates the session.

Thread Safety:
This class is not thread-safe because it's mutable. However, dedicated instance of struts action will be created by the struts framework to process the user request, so the struts actions don’t need to be thread-safe.

RecoverPasswordAction:

This action handles the password recovery. It basically gets the user from UserDAO, and persists a new PasswordRecovery into database, and then send the password recovery email to the user.

Thread Safety:
This class is not thread-safe because it's mutable. However, dedicated instance of struts action will be created by the struts framework to process the user request, so the struts actions don’t need to be thread-safe.

BasePasswordRecoveryAction:
This is the base action of actions related to password recovery. It has a method to send password recovery email.

Thread Safety:
This class is not thread-safe because it's mutable. However, dedicated instance of struts action will be created by the struts framework to process the user request, so the struts actions don’t need to be thread-safe.

ResendPasswordRecoveryEmailAction:
This action handles resending the password recovery email. It simply resends the email by calling the base class's sendPasswordRecoveryEmail().

Thread Safety:
This class is not thread-safe because it's mutable. However, dedicated instance of struts action will be created by the struts framework to process the user request, so the struts actions don’t need to be thread-safe.

ResetPasswordAction:
This action handles resetting the password of a user. It basically gets the PasswordRecovery from database, checks if it's not expired, uses RandomStringGenerator to generate a new password for the user, calls PrincipalMgrRemote to save the new password, and finally automatically logs the user in.

Thread Safety:
This class is not thread-safe because it's mutable. However, dedicated instance of struts action will be created by the struts framework to process the user request, so the struts actions don’t need to be thread-safe.

BasePasswordAction:
This is the base action of all password-related actions of this component. It simply has a passwordRecoveryDAO for subclasses to use.

Thread Safety:
This class is not thread-safe because it's mutable. However, dedicated instance of struts action will be created by the struts framework to process the user request, so the struts actions don’t need to be thread-safe.

com.topcoder.web.reg.interceptors:
AuthorizationInterceptor:
This interceptor is responsible for checking if a user is authorized to access the requested URI. If the user is anonymous and is not authorized, he will be redirected to the login page. Otherwise if the user is not authorized, the user will be redirected to an authorization error page. It will try to get a WebAuthentication from session, and if it's not present, it will create a BasicAuthentication and put it into session. It will also put a SessionInfo into session.

Thread Safety:

This class is not thread-safe because it's mutable. However, dedicated instance of struts interceptor will be created by the struts framework to process the user request, so the struts interceptor doesn’t need to be thread-safe.

ThrottleInterceptor:

This interceptor is responsible for checking if a user hits the site too frequently and should be throttled

Thread Safety:
This class is not thread-safe because it's mutable. However, dedicated instance of struts interceptor will be created by the struts framework to process the user request, so the struts interceptor doesn’t need to be thread-safe.

BaseInterceptor:

This is the base class for the interceptors of this component. It simply provides a createWebAuthentication() method.
Thread Safety:
This class is not thread-safe because it's mutable. However, dedicated instance of struts interceptor will be created by the struts framework to process the user request, so the struts interceptor don’t need to be thread-safe.

1.5 Component Exception Definitions

1.5.1 Custom exceptions

BasicActionException:

This is thrown if any error occurs in any action of this component. It's thrown by LoginAction, LogoutAction, ResendPasswordRecoveryEmailAction, RecoverPasswordAction, BasePasswordRecoveryAction, ResetPasswordAction.

Thread Safety:
This class is not thread-safe because the base class is not thread-safe.

BasicStrutsActionsConfigurationException:

This is thrown for any configuration error that occurs in this component.

Thread Safety:
This class is not thread-safe because the base class is not thread-safe.

AuthorizationInterceptorException:

This is thrown if any error occurs in AuthorizationInterceptor. It's thrown by AuthorizationInterceptor.

Thread Safety:
This class is not thread-safe because the base class is not thread-safe.

1.5.2 System exceptions

IllegalArgumentException:

This will be used to signal that an input parameter to the component is illegal.

1.6 Thread Safety

Although individual classes in this component are not thread-safe as per 1.4, this component is effectively thread-safe to use under Struts framework because dedicated instances of struts actions and interceptors will be created by the Struts framework to process each user request.

2. Environment Requirements

2.1 Environment

Development language: Java1.6 J2EE 1.5

Compile target: Java1.6 J2EE1.5

2.2 TopCoder Software Components

Logging Wrapper 2.0 – provides logging service

Base Exception 2.0 – used as the base for all custom exception classes.

Email Engine 3.2 – used to send email

Document Generator 3.1 - used to apply email template

Random String Generator 1.0 – used to generate new password

Custom Component:

User Profile and Audit Back End 1.0 – provides UserDAO and AuditDAO.

2.3 Existing Module

The existing TopCoder website code.

2.4 Third Party Components

Spring 2.5.6 (http://www.springsource.org/download)

Struts 2.2.3 (http://struts.apache.org/download.cgi)

3. Installation and Configuration

3.1 Configuration Parameters

Configuration for BaseAction:

Name	Description	Value
auditDAO	The AuditDAO instance used to perform auditing.	com.topcoder.web.common.dao.AuditDAO instance. It cannot be null. Required.
webAuthenticationSessionKey	The session key for WebAuthentication that is put into session.	java.lang.String instance. It cannot be null or empty. Optional. Default to "webAuthentication"
operationType	The operation type of this action. This is configured for each subclass for auditing purpose.	java.lang.String instance. It cannot be null or empty. Required.

Note that for LoginAction, the operationType should be configured to “login”.

Configuration for BasePasswordAction:

Name	Description	Value
passwordRecoveryDAO	The PasswordRecoveryDAO instance used to perform persistence operation on PasswordRecovery.	com.topcoder.web.common.dao.PasswordRecoveryDAO instance. It cannot be null. Required.

Please see the configuration of its base class BaseUserDAOAwareAction for more configuration parameters.

Configuration for BasePasswordRecoveryAction:

Name	Description	Value
emailSubject	The email subject of the	java.lang.String instance. It

Name	Description	Value
	password recovery email.	cannot be null but can be empty. Required.
emailBodyTemplateFilePath	The file path of the email body template. The template should be in the format of Document Generator component. The template must contain two variables “handle” and “link”.	java.lang.String instance. It cannot be null or empty. Required.
emailFromAddress	The email from address of the password recovery email.	java.lang.String instance. It cannot be null or empty. Required.
resetPasswordLinkTemplate	The template of the link for resetting password.	java.lang.String instance. It cannot be null or empty. It must have two variables '%passwordRecoveryId%' and '%hashCode%' in its content, and the content must have 2 http parameters "passwordRecoveryId=%passwordRecoveryId%" and "hashCode=%hashCode%". Required.

Please see the configuration of its base class BasePasswordAction for more configuration parameters.

Configuration for BaseUserDAOAwareAction:

Name	Description	Value
userDAO	The UserDAO instance used to perform persistence operation on User.	com.topcoder.web.common.dao.UserDAO instance. It cannot be null. Required.

Please see the configuration of its base class BaseAction for more configuration parameters.

Configuration for LoginAction:

Name	Description	Value
activeEmailStatus	The status code representing active email.	int. It can be any value. Optional. Default to 1
loginRemote	The EJB used for logging in the user.	com.topcoder.security.login.LoginRemote instance. It cannot be null. Required.
lastRequestedURIParameterName	The name of the http parameter that represents the URI the user last requested.	java.lang.String instance. It cannot be null or empty. Required.

Please see the configuration of its base class BaseUserDAOAwareAction for more configuration parameters.

Configuration for RecoverPasswordAction:

Name	Description	Value
passwordRecoveryExpiration	The number of minutes that the password recovery will expire in.	int. It must be non-negative. Required.

Please see the configuration of its base class BasePasswordRecoveryAction for more configuration parameters.

Configuration for ResendPasswordRecoveryEmailAction:

Name	Description	Value

Please see the configuration of its base class BasePasswordRecoveryAction for more configuration parameters.

Configuration for ResetPasswordAction:

Name	Description	Value
minimalPasswordLength	The minimal password length.	int. It must be non-negative. Required.
maximalPasswordLength	The maximal password length.	int. It must be non-negative. Required.
principalMgrRemote	The EJB used for editing the password.	com.topcoder.security.admin.PrincipalMgrRemote instance. It cannot be null. Required.

Please see the configuration of its base class BasePasswordAction for more configuration parameters.

Configuration for AuthorizationInterceptor:

Name	Description	Value
sessionInfoKey	The key that is used for setting SessionInfo into session.	java.lang.String instance. It cannot be null or empty. Optional. Default to "sessionInfo"
webAuthenticationSessionKey	The session key for WebAuthentication that is put into session.	java.lang.String instance. It cannot be null or empty. Optional. Default to "webAuthentication"

Configuration for ThrottleInterceptor:

Name	Description	Value
throttleEnabled	The flag indicating whether the throttle checking is enabled.	boolean. Must be true or false. Optional. Default to false
throttleMaxHits	Maximum number of hits a user can do within an interval before they start getting throttled.	int. It must be non-negative. Optional. Default to 10
throttleInterval	Length of a throttle interval in milliseconds.	int. It must be non-negative. Optional. Default to 5000

The above tables' configuration should be done in Spring XML. The attached "applicationContext.xml" is a sample Spring file.

3.2 Dependencies Configuration

Logging Wrapper should be configured properly. The depended EJB and DAO should be configured properly.

3.3 Package Structure

com.topcoder.web.reg.actions.basic
com.topcoder.web.reg.interceptors
com.topcoder.web.reg

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

Please see the demo.

4.3 Demo

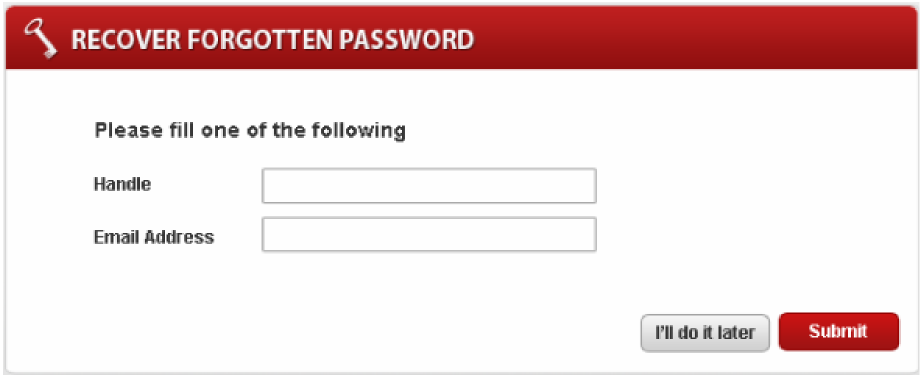
Demo API

```
// Load the Spring context file
        ClassPathXmlApplicationContext applicationContext = new ClassPathXmlApplicationContext(
            "applicationContext.xml");

        // Get the LoginAction from Spring
        LoginAction loginAction = (LoginAction) applicationContext.getBean("loginAction");

        // Log in
        loginAction.setUsername("username");
        loginAction.setPassword("password");

        // The return string should be "firstTimeLogin"
        loginAction.execute();

        // Get the ResetPasswordAction from Spring
        ResetPasswordAction resetPasswordAction = (ResetPasswordAction) applicationContext
                        .getBean("resetPasswordAction");

        // Reset the password
        resetPasswordAction.setPasswordRecoveryId(1);

        // The return string should be "success"
        resetPasswordAction.execute();

        // Get the ResetPasswordAction from Spring
        LogoutAction logoutAction = (LogoutAction) applicationContext.getBean("logoutAction");

        // Log out
        // The return string should be "success"
        logoutAction.execute();

        // Get the RecoverPasswordAction from Spring
        RecoverPasswordAction recoverPasswordAction = (RecoverPasswordAction) applicationContext
                        .getBean("recoverPasswordAction");

        // Recover the password by handle
        recoverPasswordAction.setHandle("dok_tester");

        // The return string should be "success"
        recoverPasswordAction.execute();

        // Get the ResendPasswordRecoveryEmailAction from Spring
        ResendPasswordRecoveryEmailAction resendPasswordRecoveryEmailAction =
(ResendPasswordRecoveryEmailAction) applicationContext
                        .getBean("resendPasswordRecoveryEmailAction");

        // Resend the email
        resendPasswordRecoveryEmailAction.setPasswordRecoveryId(1);

        // The return string should be "success"

        resendPasswordRecoveryEmailAction.execute();
```

Demo Web

A sample Spring configuration file “applicationContext.xml” is attached for reference only. Since Struts2 does all the work there is nothing to show in demo in fact. Here an example of recovering password is shown.

Suppose that the user requests http://localhost/showRecoverPassword, he will see a page that looks like this:



Then the user either inputs handle or email address, and click “Submit”, which will post the form to http://localhost/recoverPassword, and will be handled by RecoverPasswordAction. As a result, a new PasswordRecovery object will be saved into database, and an email will be sent to the user:

```
Subject:
Reset password e-mail for your account on TopCoder web-site
Content:
Your user handle on TopCoder web-site is: test

You can reset the password for your account by following this hyperlink:
http://localhost/resetPassword?passwordRecoveryId=123&hashCode=abc
```

The /recoverPasswordEmailSent.jsp will be rendered to the user which shows the following information:

```
Recover forgotten password has been sent into your e-mail.
Please check your email to recover your password.
If you don’t receive recovery password e-mail, Click here to send it again.
```

The above example assumes that the id of the PasswordRecovery is 123 and its hash value is “abc”, and the user handle is “test”.

If the user didn’t receive the email, he can click the above link to receive the email again. This will trigger ResendPasswordRecoveryEmailAction to send the same email again.

Upon receiving the email, the user clicks the link in the email, which will allow ResetPasswordAction to generate a new password. Then /resetPasswordConfirmation.jsp will be rendered to the user to inform him the new password.

The LoginAction and LogoutAction are similar but simpler in usage and are not shown in this demo.

5. Future Enhancements

None