Complete Profile Struts Actions v1.0 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 provides struts actions for user to complete profile data for specific user actions on specific TC Website.

1.1 Design Patterns

1.1.1 Strategy

CompleteProfileAction uses UserDAO, AuditDAO, UserIdRetriever, StateDAO, CountryDAO and TimeZoneDAO implementations which are pluggable.

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

1.1.2 DAO

UserDAO, AuditDAO, StateDAO, CountryDAO and TimeZoneDAO are the DAO used by this component.

1.1.3 MVC

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

1.1.4 IoC

The configuration is done through Spring injection. Therefore the Inversion of Control (IoC) pattern is used.

1.2 Industry Standards

XML
JSP
HTML
HTTP

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 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 throws CompleteProfileActionException. This exception 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 JSP

The developers should implement the following JSP according to the provided struts.xml:
completeProfileForForum.jsp – this will display and allow the user to edit the profile properties of ARS 2.12.1.2
completeUserProfileForWiki.jsp – this will display and allow the user to edit the profile properties of ARS 2.13.1.2
completeUserProfileForTCDirect.jsp – this will display and allow the user to edit the profile properties of ARS 2.24.1.2
completeUserProfileForOnlineReviewForCompetitor.jsp – this will display and allow the user to edit the profile properties of ARS 2.25.1.2
completeUserProfileForOnlineReviewForCustomer.jsp – this will display and allow the user to edit the profile properties of ARS 2.25.1.3
completeUserProfileForJiraForCompetitor.jsp – this will display and allow the user to edit the profile properties of ARS 2.26.1.2
completeUserProfileForJiraForCustomer.jsp – this will display and allow the user to edit the profile properties of ARS 2.26.1.3
completeUserProfileForSVNAccessForCompetitor.jsp – this will display and allow the user to edit the profile properties of ARS 2.27.1.2
completeUserProfileForSVNAccessForCustomer.jsp – this will display and allow the user to edit the profile properties of ARS 2.27.1.3

completeUserProfileForVMAccessForCompetitor.jsp – this will display and allow the user to edit the profile properties of ARS 2.28.1.2
completeUserProfileForVMAccessForCustomer.jsp – this will display and allow the user to edit the profile properties of ARS 2.28.1.3
completeUserProfileForCompetitionParticipation.jsp – this will display and allow the user to edit the profile properties of ARS 2.29.1.2
completeUserProfileForCopilotContest.jsp – this will display and allow the user to edit the profile properties of ARS 2.30.1.2
completeUserProfileForStudioForCompetitor.jsp – this will display and allow the user to edit the same profile properties as in ARS 2.29.1.2
completeUserProfileForStudioForCustomer.jsp – this will display and allow the user to edit the same profile properties as in ARS 2.27.1.3

1.3.4 Validation

XML validation is used for validations. Some input property such as “Phone Number” requires further programmatic validation. Such validation is done in CompleteProfileAction.validate(). Note that the XML validation should only apply to the saveUserProfile() method. Therefore the validation rule should be specified in the files with name “ActionName-alias-validation.xml” where “Action alias” refers to each action name with method="saveUserProfile" as given in the Struts configuration, rather than the action class name.

Because there are a lot of action aliases, this design only gives instructions on how to write the validation XML according to the rules in ARS (Application Requirement Specification), and the developers should write the actual validation XML based on instructions given here (the full tutorial of the Struts2 XML validation can be found here).

For input properties that has “Y” in “R?” column in ARS, use “requiredstring” validator like the following example:

```
<field-validator type="requiredstring">
    <param name="trim">true</param>
    <message>${getText("username.missing")}</message>
</field-validator>
```

For checking the maximal length of a string property, use “stringlength” validator like the following example:

```
<field-validator type="stringlength">
    <param name="maxLength">20</param>
    <message>${getText("username.length")}</message>
</field-validator>
```

For “Postal Code”, in addition to the “stringlength” and “requiredstring” validators, the “regex” validator should be used with the following regular expression:

```
<field-validator type="regex">
  <param name="expression">[0-9]*</param>
  <message>${getText("...")}</message>
</field-validator>
```

For “Time Zone”, in addition to the “stringlength” and “requiredstring” validators, the “regex” validator should be used with the following regular expression:

```
<field-validator type="regex">
  <param name="expression">GMT(+|-)[0-9][0-9]</param>
  <message>${getText("...")}</message>
</field-validator>
```

For “Phone Number”, the XML validation doesn’t have to handle the checking of its format because this is checked programmatically. However, “stringlength” and “requiredstring” validators should still be used.

The above discussion already covers all cases of the input properties of ARS. The following sections list which sections of ARS corresponds to which action alias in the provided struts.xml. Note that the “Data Element” of the tables of ARS can be easily matched to the properties of CompleteProfileAction because the property names are just the concatenation of the content of “Data Element” column under Java naming convention. For example, the “First Name” in ARS table corresponds to the property CompleteProfileAction.firstName, and “Current Address 1” corresponds to CompleteProfileAction.currentAddress1. The only exception is that “Student / Professional” corresponds to CompleteProfileAction.coderTypeId. The validation of CompleteProfileAction.coderTypeId is also different from the ARS. The “conversion” validator should be used to check that it’s a valid integer.

1.3.4.1 completeProfileForForum

Please see the table of ARS 2.12.1.2 for its validation rules.

1.3.4.2 completeProfileForWiki

Please see the table of ARS 2.13.1.2 for its validation rules.

1.3.4.3 completeProfileForTCDirect

Please see the table of ARS 2.24.1.2 for its validation rules.

1.3.4.4 completeProfileForOnlineReviewForCompetitor

Please see the table of ARS 2.25.1.2 for its validation rules.

1.3.4.5 completeProfileForOnlineReviewForCustomer

Please see the table of ARS 2.25.1.3 for its validation rules.

1.3.4.6 completeProfileForJiraForCompetitor

Please see the table of ARS 2.26.1.2 for its validation rules.

1.3.4.7 completeProfileForJiraForCustomer

Please see the table of ARS 2.26.1.3 for its validation rules.

1.3.4.8 completeProfileForSVNAccessForCompetitor

Please see the table of ARS 2.27.1.2 for its validation rules.

1.3.4.9 completeProfileForSVNAccessForCustomer

Please see the table of ARS 2.27.1.3 for its validation rules.

1.3.4.10 completeProfileForVMAccessForCompetitor

Please see the table of ARS 2.28.1.2 for its validation rules.

1.3.4.11 completeProfileForVMAccessForCustomer

Please see the table of ARS 2.28.1.3 for its validation rules.

1.3.4.12 completeProfileForCompetitionParticipation

Please see the table of ARS 2.29.1.2 for its validation rules.

1.3.4.13 completeProfileForCopilotContest

Please see the table of ARS 2.30.1.2 for its validation rules.

1.3.4.14 completeProfileForStudioForCompetitor

The validation rules of this action are the same as ARS 2.29.1.2.

1.3.4.15 completeProfileForStudioForCustomer

The validation rules of this action are the same as ARS 2.27.1.3.

1.3.5 Message Key and Resource Bundle

Since the developers are responsible for writing the actual XML validation files, they are also responsible for defining the message key for each validation error message, as well as the resource bundle that provides the message text.

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. These methods don’t need to be added with the javax.annotation.PostConstruct.

1.4 Component Class Overview

com.topcoder.web.reg.profilecompleteness.actions:

CompleteProfileAction:

This is the action that is responsible for getting and saving user profile for certain purpose (such as for forum participation). It extends ActionSupport and implement ServletRequestAware and ServletResponseAware. It performs the full logic of getting user profile and updating user profile. There is no need to define any subclasses for specific purpose such as for forum participation. The struts validation framework can be configured so that different rules apply to different action URL, which are all backed by this class.

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.

1.5 Component Exception Definitions

1.5.1 Custom exceptions

CompleteProfileActionConfigurationException:

This is thrown for any configuration error in this component.

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

CompleteProfileActionException:

This is thrown if any error occurs in any action of this component. It's thrown by CompleteProfileAction.

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 will be created by the Struts framework to process each user request, and because the struts action instances will not be changed after initialization, and finally the DAO interfaces and UserIdRetriever used by the actions are all thread-safe to use.

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.

Custom Component:

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

Check Profile Completeness Servlet Filter 1.0 – provides UserIdRetriever

2.3 Existing Module

The existing TopCoder website code.

2.4 Third Party Components

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

Struts 2.2.1.1 (http://struts.apache.org/download.cgi)

3. Installation and Configuration

3.1 Configuration Parameters

Configuration for CompleteProfileAction:

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.
auditDAO	The AuditDAO instance used to perform auditing.	com.topcoder.web.common.dao.AuditDAO instance. It cannot be null. Required.
userIdRetriever	The UserIdRetriever used for retrieving user id.	com.topcoder.web.reg.profilecompleteness.filter.impl.UserIdRetriever instance. It cannot be null. Required.
stateDAO	The StateDAO instance used to perform persistence operation on State.	com.topcoder.web.common.dao.StateDAO instance. It cannot be null. Required.
countryDAO	The CountryDAO instance used to perform persistence operation on Country.	com.topcoder.web.common.dao.CountryDAO instance. It cannot be null. Required.
timeZoneDAO	The TimeZoneDAO instance used to perform persistence operation on TimeZone.	com.topcoder.web.common.dao.TimeZoneDAO instance. It cannot be null. Required.
coderTypeDAO	The CoderTypeDAO instance used to perform persistence operation on CoderType.	com.topcoder.web.common.dao.CoderTypeDAO instance. It cannot be null. Required.
demographicQuestionDAO	The DemographicQuestionDAO instance used to perform persistence operation on DemographicQuestion.	com.topcoder.web.common.dao.DemographicQuestionDAO instance. It cannot be null. Required.
phoneTypeId	The phone type id that is used to set the property of phoneTypeId of the Phone entity of the user profile.	int. It must be non-negative. Required.
primary	This flag denotes whether the user phone is the primary phone.	boolean. Must be true or false. Required.

Name	Description	Value
securityKeyTypeId	The security key type id that is used to set the property of securityKeyTypeId of the UserSecurityKey entity of the user.	int. It must be non-negative. Required.
getUserProfileOperationType	The operation type for the action of getting the user profile. This is for auditing purpose.	java.lang.String instance. It cannot be null or empty. Required.
incomingRequestURIKey	The session key for retrieving the incoming request URI.	java.lang.String instance. It cannot be null but can be empty. Required.
saveUserProfileOperationType	The operation type for the action of saving the user profile. This is for auditing purpose.	java.lang.String instance. It cannot be null or empty. Required.
genderQuestionId	The id of the DemographicQuestion that asks about the gender of the user.	long. It must be non-negative. Required.
ageQuestionId	The id of the DemographicQuestion that asks about the age of the user.	long. It must be non-negative. Required.

The above tables' configuration should be done in Spring XML.

3.2 Dependencies Configuration

Logging Wrapper should be configured properly. The depended DAOs and UserIdRetriever should be configured properly.

3.3 Package Structure

com.topcoder.web.reg.profilecompleteness.actions

4. Usage Notes

4.1 Required steps to test the component

1. Configure libraries paths in **build-dependencies.xml.**

2. Execute **ant demo_war.**

3. Deploy demo.war into Tomcat server.

4. Go to http://localhost:8080/demo/ and see what you can edit.

NOTE: In http://localhost:8080/demo/getUserProfileAllFields all fields are required, except address2-3, province and state. It's used to describe all fields constraints.

4.2 Required steps to use the component

Please see the demo.

4.3 Demo

Since Struts2 does all the work there is nothing to show in demo in fact. Here an example of displaying and editing the user profile for Online Review for a customer:

Suppose the user access http://localhost/getUserProfileForOnlineReview, UserRoleCheckingAction will be called to check the user's role according to struts.xml. Because the user is a customer, this action will return "customer", which will redirect the request to "getUserProfileForOnlineReviewForCustomer" action. This action is handled by

CompleteProfileAction.getUserProfile(), which will get the User object and forward to completeProfileForOnlineReviewForCustomer.jsp. This page will display a form with properties given in ARS 2.25.1.3.

After the user edits the profile and click the button to submit the changes, the http://localhost/completeProfileForOnlineReviewForCustomer will be called by completeProfileForOnlineReviewForCustomer.jsp to submit the form, and the new user profile will be saved into database. Then the user will be redirected to a URI that is retrieved from the session under key CompleteProfileAction.incomingRequestURIKey.

5. Future Enhancements

None