# Struts Actions 1 Requirements Specification

## 1. Scope

### 1.1 Overview

We are going to move Direct(www.topcoder.com/direct) from flex to HTML based platform. Release 1 contains Launch Contest page(s). Clients can use this page to create/update studio (nonsoftware) and software contests. Existing back end services will be used.

#### *1.1.1 Version*

1.0

### 1.2 Logic Requirements

Each of the following actions will be designed as a Struts action with AJAX handling capability. It is up to the designer to make best use of Struts capabilities, including interceptors and base action classes. The only client of this application will be the JSPs, so there are no API or structural requirements. The designer is free to refactor any interceptor or action as long as requirements are met.

This component is responsible for several actions that are used to perform requests from the Frontend. The ARS sections mapping to these actions are stated.

#### *1.2.1 Struts Actions*

Here we have some general information about the actions:

- All actions will extend the AbstractAction class from the struts framework component.
- Javabean properties (i.e. getXXX/setXXX) should be used for all action data. The input data will be provided as request parameters and will need to be mapped to these setters using JavaBean conventions. The ARS will state the names of the parameters, and we will assume camel case equivalents. In essence, this component will drive how the parameters are sent to the action as well as what resultant data is sent back
- The design may rely on some data conversion to take place before the action, such as converting dates, but the designer must specify what those conversions must be.
- The execute method will place all result data in the AggregateDataModel then it will set it to the action to make it available in the ValueStack. Essentially, we expect return parameters to be sent back. Use simple “return” key for the model.

#### *1.2.2 Validation*

Validation here will comprise the user input as specified in the use cases provided in the ARS for each relevant action. Validation errors would be packaged in the ValidationErrors entity with each property that fails validation placed in the ValidationErrorRecord entity. ValidationErrorRecord contains a messages array field that allows multiple validation errors to be provided per field (for example, a field could be too long and contain incorrect format).

#### *1.2.3 Save draft contest action*

This action will create/update a draft contest, as a result of the pressing of the “Save as Draft”. One of the parameters will be the contested. If it is present, then the action will update the contest, otherwise, it will create it.

The contest type parameter will specify whether this is a studio or software competition. The contest type will be “STUDIO” for studio competitions, and others for software competitions. The contest type is saved in the competition.type field, and for the studio subtype, it goes into the

competition.contestData.contestTypeId field.

This will also determine whether the action is done on a studio or software competition. Depending on this, we either use the create/updateCompetition methods, or the create/updateSoftwareCompetition methods.

This encompasses information captured for the contest in ARS 2.4-2.10, except for management of its documents.

#### 1.2.4 *Get capacity full dates (ARS 2.4.1.4, same for below sections)*

This action will get the capacity full dates from the PipelineServiceFacade.getCapacityFullDates for the given contest type.

#### 1.2.5 *Get contest action*

This action will get a contest with the given id. Use ContestServiceFacade's getContest or getSoftwareContestByProjectId based on whether this is a studio or non-studio contest.

#### 1.2.6 *Create project action (2.4.1.6)*

This action will create a new project and return it. Use ProjectServiceFacade's createProject method.

#### 1.2.7 *Get all projects action (2.4.1.5)*

This action will get all existing projects. Use ProjectServiceFacade's getAllProjects method

#### 1.2.8 *Get all billing projects action (2.4.1.7)*

This action will get all billing projects, but just their ID, name, and description fields, so it may be selected). Use ProjectServiceFacade's getClientProjectsByUser method.

#### 1.2.9 *Struts action mapping*

The designer does not need to provide a mapping file for struts actions. All struts and spring files will be provided by the assembly.

#### 1.2.10 *Services*

The actions will directly use façade classes. These will be injected.

#### 1.2.11 *Logging and Error Handling*

The actions will not perform logging and any error handing. If any errors occur, they actions will simply put the Exceptions into the model as a "result". There will an interceptor that will process the logging of this.

#### 1.2.12 *Transaction handling*

Any transactions will be handled externally by the container at the level of the façade. Nothing needs to be done in this component.

#### 1.2.13 *Thread-safety*

Since we are operating in a servlet container, threading is not an issue.

#### 1.2.14 *Configuration*

All configuration will be done via method injection. Each item being injected via a setter will have a corresponding getter.

### 1.3 Required Algorithms

None

### 1.4 Example of the Software Usage

The actions will handle contest launching requests.

### 1.5 Future Component Direction

None

## 2. Interface Requirements

#### 2.1.1 *Graphical User Interface Requirements*

None

#### 2.1.2 *External Interfaces*

None

#### 2.1.3 *Environment Requirements*

- Development language: Java1.5, J2EE 1.5
- Compile target: Java1.5, J2EE 1.5
- Application Server: JBoss 4.0.2
- Informix 11

#### 2.1.4 *Package Structure*

com.topcoder.service.actions

## 3. Software Requirements

### 3.1 Administration Requirements

#### 3.1.1 *What elements of the application need to be configurable?*

- User session key
- Login page name
- Service Facades

### 3.2 Technical Constraints

#### 3.2.1 *Are there particular frameworks or standards that are required?*

- Struts 2.1.1
- Spring 3.0
- EJB 3.0

#### 3.2.2 *TopCoder Software Component Dependencies:*

- Struts Framework 1.0
- Contest Service Façade 1.0

- Pipeline Service Façade 1.0
- Project Service Façade 1.0

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

*3.2.3 Third Party Component, Library, or Product Dependencies:*

None

*3.2.4 QA Environment:*

- Java 1.5/J2EE 1.5
- JBoss 4.0.2
- Informix 11
- MySQL 5.1
- Struts 2.1.8.1
- Spring 3.0
- Javascript 1.8
- Mozilla Firefox 2.0/3.0
- IE 6.0/7.0
- Google Chrome
- Safari 3/4

### 3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines.

### 3.4 Required Documentation

*3.4.1 Design Documentation*

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

*3.4.2 Help / User Documentation*

- Design documents must clearly define intended component usage in the ‘Documentation’ tab of the TopCoder UML Tool.