# Struts Result JSON Serializer 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

This component provides the framework for handling user requests and integrating them with the services. The framework will be built on top of Struts 2.0. The user requests can be synchronous or asynchronous (AJAX), and this framework will be able to handle both, although its first incarnation, it will rely solely on AJAX processing.

#### *1.2.1 Result and JSON Serialization*

JSON data will be nothing more than an exact representation of the aggregate model. We will use the Json-lib (http://json-lib.sourceforge.net/) for this purpose. This component will provide an implementation of the com.opensymphony.xwork.Result interface that will be able to serialize all the variations of the data in the AggregateDataModel generated in this component. The data to be serialized will always be under the “result” key.

This includes deep instances of:

- StudioCompetition
- SoftwareCompetition
- List<Category>
- List<Technology>
- ValidationErrors
- Exception (only the message)
- Project
- ProjectData
- List<UploadedDocument> (without the content bytes)
- List<CompDocument> (without the content bytes)
- List<CapacityData>

The JSON message structure is ultimately up to the designer, as it will be used in the assembly. Something like this can work

```
{“result” :
        {
        // the result
        }
}
```

For the result, and:

```
{“error” :
        {
        // the error message
```

```
        }
    }
```

For the error message.

#### 1.2.2 *Thread-safety*

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

#### 1.2.3 *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

Data being returned to the calling application must be serialized as JSON.

### 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.ajax

## 3. Software Requirements

### 3.1 Administration Requirements

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

None

### 3.2 Technical Constraints

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

- Struts 2.1.1

- Spring 3.0

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

None

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

Json-lib (http://json-lib.sourceforge.net/)

#### 3.2.4 *QA Environment:*

- Java 1.5/J2EE 1.5
- JBoss 4.0.2
- Struts 2.1.8.1
- Spring 3.0

### 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.