Direct Struts Actions 1 1.0 Component Specification

1. Design

The design implements the actions described in the Requirement Specifications. It provides a class for every action. The actions are usual Struts 2 actions: they contains the setters and getters for the http parameters and the execute method where the business logic is performed. The facade services are injected externally: they are used to persist or retrieve the topcoder entities. An interceptor is provided to re-use the annotation validation of Struts 2 and the validation classes of Struts Framework topcoder component. A converter is also provided to permit to use all the data of the topcoder entities.

1.1 Design Patterns

The patterns used in this design are:

1. **Template method**: the BaseDirectStrutsAction delegate the business logic to the abstract executeAction method implemented by the sub classes
2. **Strategy**: the actions use an interface as business facade
3. **MVC**: the component implements the MVC pattern: the actions contain the model (the parameters) and the controller part (the business logic in the action)

1.2 Industry Standards

Xml

1.3 Required Algorithms

1.3.1 Validation

The validation is performed by the usual annotations of Struts 2. The ValidationErrorsInterceptor will put the validation messages of Struts 2 into the Validation errors.

The Struts 2 annotation validation requires a default message and an optional key.

Use an explicit default message that contains the meaning of the error. For example if the field is required and it must be a string with length max of 5 characters and it's set to 10 characters the create the message as " The *field*'s length must be not empty and max 10 characters" where *field* is the name of field. The messages are up to developer. Set also the optional key so the message can be changed without re-compile the project. The key must be constructed as: 'i18n.nameOfTheAction.nameOfFieldMeaningOfValidator". For example in the

previous case create the key 'i18n.saveDraftContestAction.contestNameRange'.

1.4 Component Class Overview

DateAfterCurrentDateValidator

This validator validates the date which must be after current date. It's used for example to validate the start date of contest.

It's used together with the CustomValidator with the 'dateAfterCurrentDate' name type. See http://java-x.blogspot.com/2008/04/struts-2-custom-validators.htmlhttp://java-x.blogspot.com/2008/04/struts-2-custom-validators.html for an example of configuration annotation.

BaseDirectStrutsAction

This is the base action of all actions in this component. It sets the AggregateDataModel to the model using the prepare logic of struts framework (in the assembly/development the logic of this class could be implemented directly in AbstractAction and delegate the logic of the action to the template method

It also manages the exceptions thrown by the template methods

GetContestAction

This action will get a contest with the given id. It uses ContestServiceFacadeÔs getContest or getSoftwareContestByProjectId based on whether this is a studio or non-studio contest (based on the presence of the ids)

ContestAction

This is a base class only to hold the contest service facade shared among several actions

GetCapacityFullDatesAction

This action will get the capacity full dates from the PipelineServiceFacade.getCapacityFullDates for the given contest type. See here http://forums.topcoder.com/?module=Thread&threadID=666827&start=0 for further details.

ProjectAction

This is a base class only to hold the project facade shared among several actions

GetAllBillingProjectsAction

This action will get all billing projects, but just their ID, name, and description fields. See (2.4.1.7) in ARS

GetAllProjectsAction

This action will get all existing projects. It uses ProjectServiceFacadeÕs getAllProjects method. See 2.4.1.5 of ARS.

ValidationErrorsInterceptor

This purpose of this interceptor is to adapt the default validation workflow of Struts 2 to the validation workflow of the Struts Framework TC component.

It retrieves the validation errors of fields (also called properties in Struts Framework) provided by the validation annotationsand fills the ValidationErrors with the errors messages. In this mode the component can completely re-use all validation annotations provided by Struts 2

CreateProjectAction

This action permits to create a project. See (2.4.1.6) of ARS. It simply collect the data to create a project and then create it.

XMLGregorianCalendarTypeConverter

This is the converter for the XMLGregorianCalendar type. This data type is intensively used in the project and contest entities so it's necessary to convert the data from and to this data type

SaveDraftContestAction

This action permit to create or update a contest. The contest could be a studio or a software competition, based on competition type. The action is an update based on the the project id or contest id presence. This class action contains also in the prepare method another action: it permits to write the sub-entities of the software and/or studio

competition so two http round trips will be provided in this action. The action validates only a subset of fields, check here in the forum the required fields: http://forums.topcoder.com/?module=Thread&threadID=666815&start=0 .

1.5 Component Exception Definitions

There are no exceptions defined. The exceptions are re-used and set to the model if they occur (see BaseDirectStrutsAction class)

1.6 Thread Safety

The custom validator and the custom converter classes are thread safe because they are stateless and the parent classes are thread safe. The actions are not thread safe but the thread safety is not necessary because in Struts 2 the actions are created in every request: so they can be mutable, they are performed by a single thread. Concluding, the component is thread safe.

2. Environment Requirements

2.1 Environment

2.1.1 Java 1.5/J2EE 1.5
2.1.2 J Boss 4.0.2
2.1.3 Informix 11
2.1.4 MySQL 5.1
2.1.5 Struts 2.1.8.1
2.1.6 Spring 3.0
2.1.7 Javascript 1.8
2.1.8 Mozilla Firefox 2.0/3.0
2.1.9 IE 6.0/7.0
2.1.10 Google Chrome
2.1.11 Safari 3/4

2.2 TopCoder Software Components

2.2.1 Struts Framework 1.0 : base framework classes
Contest Service Façade 1.0: used to retrieve/manage the contests
2.2.2 Pipeline Service Façade 1.0: used to manage the pipeline
Project Service Façade 1.0: used to manage the projects

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

2.3.1 Struts 2.1.8

3. Installation and Configuration

3.1 Package Name

com.topcoder.service.actions

(It would be possible to move the converter, the validator and the interceptor to other packages but not under actions package, under directly service package for example service.interceptors like in the Struts Framework 1 component. This permission needs the architect after the final fix)

3.2 Configuration Parameters

The facades and other properties of the action must be injected externally. There is not an unique name for these configuration parameters, they can be injected using annotations (in the assembly) or Spring configuration files.

3.3 Dependencies Configuration

The topcoder components and Struts 2 framework must be configured properly.

4. Usage Notes

None

4.1 Required steps to test the component

Extract the component distribution.

Execute ‘ ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

See the demo section

4.3 Demo

Firstly I setup the conversion properties to setup the xml gregorian calendar converter:

```
# here I define the XMLGregorianCalender converter in the xwork-conversion.properties
  javax.xml.datatype.XMLGregorianCalendar = com.topcoder.service.actions.XMLGregorianCalendarTypeConverter
```

Then I setup the validator in the validators.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator Config 1.0//EN"
"http://www.opensymphony.com/xwork/xwork-validator-config-1.0.dtd">
<!-- I define the date after current date validator  in the validator.xml , then I can use it with the
@CustomValidator annotation-->
<validators>
  <validator name="dateAfterCurrentDateValidator"
class="com.topcoder.service.actions.DateAfterCurrentDateValidator"/>
</validators>
```

At this point I set up the struts.xml to configure the action and the interceptor:

```
<!DOCTYPE struts PUBLIC
      "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
      "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

        <package name="tcs-default" extends="struts-default" abstract="true">
                <!-- Setup interceptors stack -->
                <interceptors>
                        <interceptor name="authentication" class="authenticationInterceptor" />
                        <interceptor name="logging" class="loggingInterceptor" />
                        <!-- I define here the validation error interceptor -->
                        <interceptor name="validationErrors"
class="com.topcoder.service.actions.ValidationErrorsInterceptor" />

                        <interceptor-stack name="defaultTCSStack">
                                <interceptor-ref name="authentication" />
                                <interceptor-ref name="logging" />
                                <interceptor-ref name="validationErrors" />
                                <interceptor-ref name="defaultStack" />
                        </interceptor-stack>
                </interceptors>

                <!-- Make the default one used for all actions unless otherwise configured. -->
                <default-interceptor-ref name="defaultTCSStack" />

                <!-- Configure global results for AuthenticationInterceptor -->
                <global-results>
                        <result name="login">/example/Login.jsp</result>

                </global-results>
        </package>

        <package name="default" namespace="/" extends="tcs-default">

                <!-- login action -->
                <action name="createProject" class="com.topcoder.service.actions.CreateProjectAction">
                        <result name="success">projectCreated</result>
                </action>

        </package>

</struts>
```

At the end I can create the related jsp page for creating the project:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
```

```
<head>
<title>Create a project</title>
</head>
<body>
<!-- The first http round trip to prepare the jsp data is made by another action in another component -->
<table align=center>
        <tr>
                <th>Write project name </th>
                <th>Write description of project</th>
        </tr>


        <tr>
                <td class="nowrap"><s:textField label="Project name"
                name="projectName"/></td>
                <td class="nowrap"><s:textArea label="Project description"
                        name="projectDescription"/></td>
        </tr>


</table>
</body>
</html>
```

5. Future Enhancements

It's possible to implement other actions.