[TopCoder]

Struts Framework 1.0 Component Specification

1. Design

We are going to move Direct(www.topcoder.com/direct) from flex to HTML based platform. This contest is a architecture contest that will mainly focus on defines the front end technical approach to meet the requirements of the project's release 1. 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.
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 AbstractAction holds the common data such as AggregateDataModel, it's expected that all the other actions should extend from this action. This component also defines two interceptors. AuthenticationInterceptor is used to check that user info has been in the session, and LoggingInterceptor is used to log the exceptions thrown by the actions.

1.1 Design Patterns

MVC – This is the foundation of this framework. The actions define in this component implement the controller part of the MVC pattern.

Interceptor – The AuthenticationInterceptor and LoggingInterceptor are used to authenticate user and log errors.

DTO- the ValidationErrors, ValidationErrorRecord and AggregateDataModel are data transfer objects.

1.2 Industry Standards

Struts, JavaBean, IoC, XML (for configuration)

1.3 Required Algorithms

All the implementation details are covered in the method document.s

1.4 Component Class Overview

AuthenticationInterceptor

This interceptor will be responsible for checking if the user info is in session. It will look for session key userSessionIdentityKey. If found, it allows the action to proceed. If not, it is sent to the login page.

The user session key and the login page will be configurable.

It's mutable and not thread safe.

LoggingInterceptor

This interceptor will simply log the exception. Basically, when the action is done, it will intercept the result, and if on the stack, there is a AggregateDataModel on the stack and in it there will be a result under key result. If its value is an Throwable, log its error message and stacktrace using the Logging Wrapper.

It's mutable and not thread safe.

AbstractAction

Represents the AbstractAction.

It holds the attributes model, and action.

It's mutable and not thread safe.

AggregateDataModel

Represents the AggregateDataModel entity.

It holds the attributes data, and action.

It's mutable and not thread safe.

ValidationErrors

Represents the ValidationErrors that holds the validation errors.

It holds the attribute errors.

It's mutable and not thread safe.

ValidationErrorRecord

Represents the ValidationErrorRecord entity.

It holds the attributes property name, and messages.

It's mutable and not thread safe.

1.5 Component Exception Definitions

1.5.1 Component Exceptions

LoggingInterceptorException

LoggingInterceptorException is thrown by LoggingInterceptor if any error is caught during the operation.

This class is not thread safe as its parent class is not thread safe.

1.5.2 System Exceptions

IllegalArgumentException

This exception is thrown if given parameter is illegal.

IllegalStateException

This exception is thrown if necessary statuses are missing.

1.6 Thread Safety

This AbstractAction is not thread safe and the implementations do not need be thread safe. This should be fine since different instances of this class should be created to serve different user requests and will not be shared across user threads. So it can be considered being used in a thread safe manner.

The AggregateDataModel(and other DTO objects) is not thread safe. This should be fine since each user has data model different instances with life-cycle bound to the request/response round-trip, thus the data model will not be shared across user threads. So it can be considered being used in a thread safe manner.

The AuthenticationInterceptor and LoggingInterceptor are technically not thread safe since they have mutable states, but those are either intent to be Ioc injected and should not change afterwards. Thus they can be considered being used in a thread safe manner.

2. Environment Requirements

2.1 Environment

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

2.2 TopCoder Software Components

BaseException 2.0.0 – used to provide the base of custom exceptions.

Logging Wrapper 2.0 – used to log errors.

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

Struts 2.1.6

Spring 2.5.6 (used for create objects and configure them by IoC injection)

3. Installation and Configuration

3.1 Package Name

com.topcoder.service.actions
com.topcoder.service.interceptors

3.2 Configuration Parameters

3.2.1 Configure Objects by Spring

This component uses Spring to create objects. To Integrate Spring & Struts:

1. Add "*struts.objectFactory = spring*" in *struts.properties* to indicate Struts to use Spring for object creation.
2. Add "*org.springframework.web.context.ContextLoaderListener*" listener class in *web.xml*.

And the objects have proper setters so that Spring can inject configured values. Following tables describe the configuration expected by this component.

Configuration of any AbstractAction implmentations:

Property Name	Property Value	Required
secured	Whether this action is secured. It is default to be true.	No
action	The action name of this action. It is default to be the concrete implementation's class name. Must be non-null and non-empty if present.	No

Configuration of AuthenticationInterceptor:

Property Name	Property Value	Required
userSessionIdentityKey	The key of user identity in session. It is default to be "USER_ID_KEY". Must be non-null and non-empty if present.	No
loginPageName	The login page name. It is default to be "login". Must be non-null and non-empty if present.	No

Configuration of AuthorizationInterceptor:

Property Name	Property Value	Required
loggerName	The logger name is used to get the logger from the LoggerManager	Yes

A sample applicationContext.xml:

```
<beans xmlns ="http://www.springframework.org/schema/beans"
   xmlns:xsi ="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation ="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.xsd" >
   <!-- AuthenticationInterceptor -->
   <bean id="authenticationInterceptor" class=" com.topcoder.service.interceptors.AuthenticationInterceptor">
      <property name="loginPageName" value="login"></property>
      <property name="userSessionIdentityKey" value="USER_ID_KEY"></property>
   </bean>

   <!-- LoggingInterceptor -->
   <bean id="loggingInterceptor" class=" com.topcoder.service.interceptors.LoggingInterceptor">
      <property name="loggerName" value="strutsLoggerName"></property>
   </bean>

   <!-- A custom action. Note the scope is "prototype". -->
   <bean id="loginAction" class="example.LoginAction" scope="prototype">
      <property name="action" value="loginAction"></property>
   </bean>
   <!--other actions can be defined here-- >

 </beans>
```

3.2.2 *Setup Interceptors Struts*

To refer a Spring created object, the “class” attribute should be same as the bean id in Spring configuration. A sample struts.xml:

```
<!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" />

         <interceptor-stack name="defaultTCSStack">
            <interceptor-ref name="authentication"/>
            <interceptor-ref name="logging"/>
            <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="login" class="loginAction">
         <result name="input">/example/Login.jsp</result>
         <result type="redirectAction">employees</result>
      </action>

      <!-- employees action -->
```

```
        <action name="employees" class="employeeAction">
            <result>/example/Employees.jsp</result>
        </action>
    </package>

</struts>
```

3.3 Dependencies Configuration

See CS of dependencies components for their configuration.

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

1. See CS 3.2 for configuration
2. Follow the demo

4.3 Demo

4.3.1 Use AbstractAction

Create an action to retrieve employees:

```
public class EmployeeAction extends AbstractAction {

    public AggregateDataModel prepareAggregateDataModel() {
            AggregateDataModel  model = new AggregateDataModel();
            // set the model attributes here
            return model;
    }
    // the other methods are ignored here

}
```

You can see CS3.2 for how the action and handler are configured. Then write a JSP page to view the employees:

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head>
    <title>Employees</title>
</head>
<body>
            <table width=600 align=center>
                <tr> <s:url id="insert" action="wizard"/>
                    <td><s:a href="%{insert}" target="_blank">Click Here to Add New Employee</s:a></td>
                </tr>
                <tr> <s:url id="refresh" action="employees"/>
                    <td><s:a href="%{refresh}">Click Here to Refresh</s:a></td>
                </tr>
            </table><br/>
<table align=center>
    <tr>
        <th>First Name</th>
        <th>Last Name</th>
        <th>Address</th>
        <th>Age</th>
        <th>Department</th>
        <th>Introducer</th>
    </tr>
    <s:iterator value="%{model.dataAsMap['Employees']}" status="status">
        <tr>
            <td class="nowrap"><s:property value="firstName"/></td>
            <td class="nowrap"><s:property value="lastName"/></td>
            <td class="nowrap"><s:property value="address"/></td>
```

```
            <td class="nowrap"><s:property value="age"/></td>
            <td class="nowrap"><s:property value="department.name"/></td>
            <td class="nowrap"><s:property value="introducer"/></td>
        </tr>
    </s:iterator>
    </table>
</body>
</html>
```

4.3.2 Use Authentication and Logging Interceptors

See CS3.2.2 for how the interceptors are setup. Based on the configured global results, if the user is not authenticated to invoke an action, he will be taken to Login.jsp; if an error is caught during the execution, the exception message and trace stack will be logged.

5. Future Enhancements

Different implementations of RequestManager could be provided.