TC Refactoring Stage 1 Contest Services 1.0 Component Specification

1. Design

Topcoder is going to start taking some steps to refactor and improve the TopCoder community sites. The first one we'll start with is www.topcoder.com/tc. The first improvements involve updating some of the "services" that provide contest data to the front end. The current structure forces the contest types to be split into separate pages (i.e. there is a separate active/past/stats/etc. page for each contest type.

This component implements the Contest Services.

1.1 Design Patterns

1.1.1 Strategy

ActiveContestsManagerAction uses ActiveContestsManager implementations which are pluggable.
CategoriesManagerAction uses CategoriesManager implementations which are pluggable.
ContestStatusManagerAction uses ContestStatusManager implementations which are pluggable.
PastContestsManagerAction uses PastContestsManager implementations which are pluggable.

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

1.1.2 DAO/DTO

ActiveContestsManager, ContestStatusManager , and PastContestsManager can be viewed as DAO for the DTO ActiveContestDTO, ContestStatusDTO, and PastContestDTO.

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
EJB
JSON
HQL

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 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 name on the application server-side (String max 50 chars, non-empty),
- Date and timestamp (String, date/time format with year/month/day and hour/minute/seconds/milliseconds, non empty),
- Method entry/exit (on the DEBUG level, serialized to String, max 4096 chars, can be empty),
- Parameters of the methods and return values (on the DEBUG level, serialized to String, max 4096 chars, can be empty),
- Stack trace for the exception (String, max 4096 chars, non empty),
- Exception name (if an exception occurred, String, max 100 chars, non empty).

1.3.2 HQL in ActiveContestsManagerImpl.retrieveActiveContests()

The pseudo code HQL for getting all active contests is given below:

```
select new map(
        ProjectGroupCategoryLookup.name as type,
       ProjectCategoryLookup.name as subType,
       ProjectCatalogLookup.name as catalog,
       projectNameInfo.value as contestName,
       count(s) as numberOfSubmissions,
        count(ResourceInfo) as numberOfRegistrants),
        registrationPhase.scheduledEndTime as registrationEndDate,
       submissionPhase.scheduledEndTime as submissionEndDate,
       firstPrizeInfo.value as firstPrize,
       reliabilityBonusInfo.value as reliabilityBonus,
       digitalRunInfo.value as digitalRunPoints,
        digitalRunFlagInfo.value as digitalRunFlag,
        paymentsInfo.value as payments)
from Project
join ProjectCategoryLookup on
Project.projectCategoryId=ProjectCategoryLookup.projectCategoryId // this join is for sub-
type

join ProjectGroupCategoryLookup on
ProjectCategoryLookup.projectGroupCategoryId=ProjectGroupCategoryLookup.projectGroupCategoryI
d // this join is for type

join ProjectCatalogLookup on
ProjectGroupCategoryLookup.projectCatalogId=ProjectCatalogLookup.projectCatalogId // this
join is for catalog

join ProjectInfo projectNameInfo on projectNameInfo.projectId=Project.projectId and
projectNameInfo.projectInfoTypeId={projectNameInfoId} // this join is for getting the contest
name

join Upload u on u.projectId=Project.projectId
join Submission s on u.uploadId=Submission.uploadId and
s.submissionTypeId={contestSubmissionTypeId} and s.submissionStatusId in
{activeSubmissionStatusId, failedScreeningSubmissionStatusId, failedReviewSubmissionStatusId,
completedWithoutWinSubmissionStatusId} // these two join gets the number of submissions

join Resource r2 on r2.projectId=ProjectId and r2.resourceRoleId={submitterRoleId} // this
join gets the number of registrants

join ProjectPhase registrationPhase on registrationPhase.projectId=Project.projectId and
registrationPhase.phaseTypeId={registrationPhaseTypeId} // this join is for getting the
registration end date

join ProjectPhase submissionPhase on submissionPhase.projectId=Project.projectId and
submissionPhase.phaseTypeId={submissionPhaseTypeId} // this join is for getting the
submission end date

join ProjectInfo firstPrizeInfo on firstPrizeInfo.projectId=Project.projectId and
firstPrizeInfo.projectInfoTypeId={firstPlaceCostInfoId} // this join is for getting the
winner prize

join ProjectInfo reliabilityBonusInfo on reliabilityBonusInfo.projectId=Project.projectId and
reliabilityBonusInfo.projectInfoTypeId={reliabilityBonusCostInfoId} // this join is for
getting the reliability bonus

join ProjectInfo digitalRunInfo on digitalRunInfo.projectId=Project.projectId and
digitalRunInfo.projectInfoTypeId={digitalRunPointInfoId} // this join is for getting the
digital run points

join ProjectInfo paymentsInfo on paymentsInfo.projectId=Project.projectId and
paymentsInfo.projectInfoTypeId={paymentsInfoId} // this join is for getting the payments

join ProjectInfo digitalRunFlagInfo on digitalRunFlagInfo.projectId=Project.projectId and
digitalRunFlagInfo.projectInfoTypeId={ digitalRunFlagInfoId} // this join is for getting the
flag denoting whether DR is on or not
```

```
where Project.projectStatusId={activeStatusId} // this condition means only active projects
are returned.
and (submissionPhase.phaseStatusId={openPhaseStatusId} or
registrationPhase.phaseStatusId={openPhaseStatusId}) // these two means only active contests
are returned
and not exist (select * from ContestEligibility where ContestEligibility.isStudio=false and
ContestEligibility.contestId=Project.projectId) // this filters out the studio contests
order by {columnName}
```

And finally depending on the value of sortingOrder, use “asc” or “desc” for the sorting.

If all the properties of the filter are null or empty, then no filtering performed and the above HQL will be used without appending further condition clauses.

If filter.types is not empty, append a condition:

```
type in (the elements of filter.types connected by ",")
```

Similarly, if filter.subTypes or catalogs are not empty, append similar IN conditions on subType and catalog.

If contestName is not null or empty, append a condition:

```
contestName LIKE {filter.contestName}
```

If registrationStartDate is not null

If intervalType is BEFORE, append:

```
registrationPhase.scheduledStartTime < {registrationStartDate.firstDate}.
```

If intervalType is AFTER, append:

```
registrationPhase.scheduledStartTime > {registrationStartDate.firstDate}.
```

If intervalType is ON, append:

```
registrationPhase.scheduledStartTime = {registrationStartDate.firstDate}
```

If intervalType is BETWEEN_DATES, append:

```
registrationPhase.scheduledStartTime between {registrationStartDate.firstDate}
and {registrationStartDate.secondDate}
```

If intervalType is BEFORE_CURRENT_DATE, append:

```
registrationPhase.scheduledStartTime < CURRENT_DATE
```

If intervalType is AFTER_CURRENT_DATE, append:

```
registrationPhase.scheduledStartTime > CURRENT_DATE
```

If submissionEndDate or contestFinalizationDate is not null, append similar conditions as above.
For submissionEndDate, use submissionEndDate column to compare.
For contestFinalizationDate, because the contest is not over yet, we can only use the Final Review Due Date as the finalization date. To do this, the HQL must be added with the following join:

```
join ProjectPhase finalReviewPhase on finalReviewPhase.projectId=Project.projectId and
finalReviewPhase.phaseTypeId={finalReviewPhaseTypeId}
```

Then use finalReviewPhase.scheduledEndTime for contestFinalizationDate.

If filter.prizeLowerBound is not Filterable.OPEN_INTERVAL, append:

```
firstPrize >= {filter.prizeLowerBound}.
```

If filter.prizeUpperBound is not Filterable.OPEN_INTERVAL, append:

```
firstPrize <= {filter.prizeUpperBound}.
```

For the above two cases, note that the HQL CAST(...as...) expression should be used to convert the string value of firstPrize into integer for conversion.

1.3.3 HQL in PastContestsManagerImpl.retrievePastContests()

The HQL in this method is very similar to the HQL of ActiveContestsManagerImpl.retrieveActiveContests(), except that the following columns (and the required join statements for these columns) are not needed:

```
registrationPhase.scheduledEndTime as registrationEndDate,
       submissionPhase.scheduledEndTime as submissionEndDate,
       firstPrizeInfo.value as firstPrize,
       reliabilityBonusInfo.value as reliabilityBonus,
       digitalRunInfo.value as digitalRunPoints
```

And the {activeStatusId} should be changed to {completedStatusId}.

In addition, some new join statements and columns should be added.
The following join provides the completionDate:

```
join ProjectPhase approvalPhase on approvalPhase.projectId=Project.projectId and
approvalPhase.phaseTypeId={approvalPhaseTypeId}
```

A new column for completionDate should be added:

```
approvalPhase.actualEndTime as completionDate.
```

The following joins will get all the submissions of the project:

```
join Resource r3 on r3.projectId=Project.projectId
join ResourceSubmission rs2 on rs2.submissionId= r3.resourceId
join Submission on Submission.submissionId=rs2.submissionId and Submission.screening_score >=
{passingScreeningScore}
```

A new column for PastContestDTO.passedScreeningCount should be added:

```
count(Submission) as passedScreeningCount
```

The following join gets the external id of the winner:

```
join ProjectInfo winnerIdInfo on winnerIdInfo.projectId=Project.projectId and
winnerIdInfo.projectInfoTypeId={winnerIdInfoId}
```

A new column for the External Reference ID of the winner should be added:

```
winnerIdInfo.value as winnerExternalReferenceId
```

The following joins get the winner score:

```
join ProjectResult on ProjectResult.projectId=Project.projectId and ProjectResult.placed=1
```

A new column for winnerScore should be added:

```
ProjectResult.finalScore as winnerScore
```

In addition to above differences, the handling of filter in this method is also slightly different from that of ActiveContestsManagerImpl.retrieveActiveContests(). There is no need to consider prizeLowerBound and prizeUpperBound, and there is a new filtering condition to take into account. If PastContestFilter.winnerHandle is not null or empty, the following joins will get the winner handle:

```
join Resource r4 on r4.projectId=Project.projectId
join ResourceInfo externalReferenceIdInfo on externalReferenceIdInfo.resourceId=r4.resourceId
and externalReferenceIdInfo.resourceInfoTypeId={externalReferenceIdInfoId} and
externalReferenceIdInfo.value=winnerExternalReferenceId
join ResourceInfo handleInfo on handleInfo.resourceId=r4.resourceId and
handleInfo.resourceInfoTypeId={handleInfoId}
```

Then a new condition should be appended to the HQL

```
handleInfo.value LIKE {filter.winnerHandle}
```

In addition, the handling of filter.contestFinalizationDate is different. For this method, the actual end date of the approval phase should be used, instead of the final review due date.
Therefore completionDate column should be used for comparison with contestFinalizationDate.

1.3.4 HQL in ContestStatusManager.retrieveContestStatuses ()

The pseudo code HQL for getting all contest status is given below:

```
select new map(
        ProjectGroupCategoryLookup.name as type,
       ProjectCategoryLookup.name as subType,
       ProjectCatalogLookup.name as catalog,
       submissionPhase.scheduledEndTime as submissionDueDate,
       finalReviewPhase.scheduledEndTime as finalReviewDueDate,
        PhaseType.name as currentPhase
        handleInfo.value as firstPlaceHandle,
        handleInfo2.value as secondPlaceHandle)
from Project
join ProjectCategoryLookup on
Project.projectCategoryId=ProjectCategoryLookup.projectCategoryId // this join is for sub-
type
join ProjectGroupCategoryLookup on
ProjectCategoryLookup.projectGroupCategoryId=ProjectGroupCategoryLookup.projectGroupCategoryI
d // this join is for type

join ProjectCatalogLookup on
ProjectGroupCategoryLookup.projectCatalogId=ProjectCatalogLookup.projectCatalogId // this
join is for catalog

join ProjectPhase submissionPhase on submissionPhase.projectId=Project.projectId and
submissionPhase.phaseTypeId={submissionPhaseTypeId} // this join is for getting the
submission due date

join ProjectPhase finalReviewPhase on finalReviewPhase.projectId=Project.projectId and
finalReviewPhase.phaseTypeId={finalReviewPhaseTypeId} // this join is for getting the final
review phase due date

join ProjectPhase on ProjectPhase.projectId=Project.projectId and
ProjectPhase.phaseStatusId={openPhaseStatusId} // this join gets the current phase

join PhaseType on ProjectPhase.phaseTypeId=PhaseType.phaseTypeId // this gets the name of the
current phase

join ProjectInfo winnerIdInfo on winnerIdInfo.projectId=Project.projectId and
winnerIdInfo.projectInfoTypeId={winnerIdInfoId} // this join gets the winner external
reference ID

join Resource r1 on r1.projectId=Project.projectId
join ResourceInfo externalReferenceIdInfo on externalReferenceIdInfo.resourceId=r1.resourceId
and externalReferenceIdInfo.resourceInfoTypeId={externalReferenceIdInfoId} and
externalReferenceIdInfo.value=winnerIdInfo.value // these joins get the Resource representing
the winner

join ResourceInfo handleInfo on handleInfo.resourceId=r1.resourceId and
handleInfo.resourceInfoTypeId={handleInfoId} // this join gets the winner handle

join ProjectInfo secondPlaceIdInfo on secondPlaceIdInfo.projectId=Project.projectId and
secondPlaceIdInfo.projectInfoTypeId={secondPlaceIdInfoId} // this join gets the second
place’s external reference ID

join Resource r2 on r2.projectId=Project.projectId
join ResourceInfo externalReferenceIdInfo on externalReferenceIdInfo.resourceId=r2.resourceId
and externalReferenceIdInfo.resourceInfoTypeId={externalReferenceIdInfoId} and
externalReferenceIdInfo.value=secondPlaceIdInfo.value // these joins get the Resource
representing the second place

join ResourceInfo handleInfo2 on handleInfo2.resourceId=r2.resourceId and
handleInfo2.resourceInfoTypeId={handleInfoId} // this join gets the winner handle
order by {columnName}
```

And finally depending on the value of sortingOrder, use “asc” or “desc” for the sorting.

The handling of filter in this method is almost the same as ActiveContestsManagerImpl.retrieveActiveContests(), except that filter.winnerHandle and contestFinalizationDate should be handled as described in 1.3.3.

1.3.5 Wildcard Handling

For all “Like” conditions in HQL, the string to compare should be surrounded with “%”. So for example,
`contestName LIKE {filter.contestName}`
should be

```
contestName LIKE “%{filter.contestName}%”
```

1.3.6 Error Page

When the actions of this component are deployed, the configuration should ensure that if any exception is thrown by these actions, the user is redirected to an error page. Since this is handled by the Struts configuration, it’s out of scope of this component.

1.3.7 JSON Format

When calling the BaseJSONParameterAction subclass actions of this component, the caller should send a JSON string as an http parameter. This JSON object should be in the following form:

```
{ columnName:”..”,
  sortingOrder:”..”,
  pageNumber: “..”,
  pageSize: “..”,
  filter: “..” // the format of this value is the standard JSON string of any one of the 3
               // filter classes
}
```

Note that “-1” is used as the value of pageNumber and pageSize to suggest returning all pages, and it’s also used as the value of BasePrizeFilter.prizeLowerBound and prizeUpperBound to suggest open interval is used.

For enums such as SortingOrder, the value in the JSON string should be the name of the enum, such as “ASCENDING”.

1.3.8 Hibernate Mapping Files

The developers should write the hibernate mapping files for the entities in com.topcoder.web.tc.impl.entity. The mapping is simple property-to-column mapping without any association. The class documentation of these classes has stated which table they are mapped to. The property name of these entities can be straight forwardly mapped to the column names of the tables. It’s just that the property name uses Java naming convention while the database column names are under database naming convention. So for example, it’s easy to see that Project.projectId should map to ‘project.project_id’.
The only exception are the following:
ProjectCategoryLookup.projectGroupCategoryId maps to project_category_lu.project_group_category_lu_project_group_category_id;
ProjectGroupCategoryLookup.projectCategoryId maps to project_group_category_lu.project_catalog_lu_project_type_id.

1.3.9 @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. See the attached “applicationContext.xml” for example. These methods don’t need to be added with the javax.annotation.PostConstruct.

1.4 Component Class Overview

com.topcoder.web.tc:
SortingOrder:
This enum is used to indicate what sorting order to use.
Thread Safety:
This class is thread-safe because it's immutable

ActiveContestsManager(interface):
This interface defines the contract for getting information of active contests by various filtering conditions. It has an overloaded method that supports pagination.

Thread Safety:

The implementation of this interface must be thread-safe after configuration and assuming that the passed in parameters are not changed by other threads at the same time.

ContestStatusManager(interface):

This interface defines the contract for getting information of contest status by various filtering conditions. It has an overloaded method that supports pagination.

Thread Safety:
The implementation of this interface must be thread-safe after configuration and assuming that the passed in parameters are not changed by other threads at the same time.

PastContestsManager(interface):

This interface defines the contract for getting information of past contests by various filtering conditions. It has an overloaded method that supports pagination.

Thread Safety:
The implementation of this interface must be thread-safe after configuration and assuming that the passed in parameters are not changed by other threads at the same time.

CategoriesManager(interface):

This interface defines the contract for getting lookup values of contest type, sub-types, and catalogs.

Thread Safety:
The implementation of this interface must be thread-safe after configuration and assuming that the passed in parameters are not changed by other threads at the same time.

com.topcoder.web.tc.action:
PastContestsManagerAction:

This action simply wraps around PastContestsManager and provides its service to the front end.

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.

CategoriesManagerAction:

This action simply wraps around CategoriesManager and provides its service to the front end.

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.

ActiveContestsManagerAction:

This action simply wraps around ActiveContestsManager and provides its service to the front end.
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.

ContestStatusManagerAction:

This action simply wraps around ContestStatusManager and provides its service to the front end.

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.

BaseJSONParameterAction:
This is the base class for the actions that have a JSON object as parameter.
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.

com.topcoder.web.tc.dto:
PastContestDTO:
This is a simple container class that has information of a past contest.

Thread Safety:
This class is not thread-safe because it's mutable

ActiveContestDTO:
This is a simple container class that has information of an active contest.
Thread Safety:
This class is not thread-safe because it's mutable

DateIntervalType:
This enum defines the options of filtering on dates, such as filtering before/after/on a certain date, between some dates, etc.
Thread Safety:
This class is thread-safe because it's immutable

ActiveContestFilter:
This class defines the filtering condition for searching active contests.

Thread Safety:
This class is not thread-safe because it's mutable

Filterable(interface):
This is a marker interface used by the caller of this component to suggest that all pages should be returned or that the open interval should be used, such as when filtering on the prize money.
Thread Safety:
This is a marker interface that has no method. So it's thread-safe.

DateIntervalSpecification:
This class is a simple container class that defines the filtering condition on dates, such as filtering on data that is between a start date and end date.
Thread Safety:
This class is not thread-safe because it's mutable

PastContestFilter:
This class defines the filtering condition for searching past contests.

Thread Safety:
This class is not thread-safe because it's mutable

BasePrizeFilter:

This class defines the common properties for filters that filter on prize.

Thread Safety:
This class is not thread-safe because it's mutable

BaseContestDTO:

This is a simple container class that defines the common properties of active contest and past contest.

Thread Safety:
This class is not thread-safe because it's mutable

BaseFilter:

This class defines the common properties of all filter classes of this component. It also implements Filterable marker interface.
Thread Safety:
This class is not thread-safe because it's mutable

ContestStatusFilter:

This class defines the filtering condition for searching contest statuses.

Thread Safety:
This class is not thread-safe because it's mutable

ContestStatusDTO:

This class is a simple container class that contains the information of the status of a contest.

Thread Safety:
This class is not thread-safe because it's mutable

com.topcoder.web.tc.impl:
ActiveContestsManagerImpl:

This is the default implementation of ActiveContestsManager. It extends HibernateDaoSupport to execute HQL query to get the data.

Thread Safety:
This class is not technically thread-safe because it's mutable. However, it can be considered effectively thread-safe after configuration assuming that it's not changed after configuration and the passed in parameters are not changed by other threads at the same time because it holds no mutable state under such assumptions

ContestStatusManagerImpl:

This is the default implementation of ContestStatusManager. It extends HibernateDaoSupport to execute HQL query to get the data.

Thread Safety:
This class is not technically thread-safe because it's mutable. However, it can be considered effectively thread-safe after configuration assuming that it's not changed after configuration and the passed in parameters are not changed by other threads at the same time because it holds no mutable state under such assumptions

PastContestsManagerImpl:

This is the default implementation of PastContestsManager. It extends HibernateDaoSupport to execute HQL query to get the data.

Thread Safety:
This class is not technically thread-safe because it's mutable. However, it can be considered effectively thread-safe after configuration assuming that it's not changed after configuration and the passed in parameters are not changed by other threads at the same time because it holds no mutable state under such assumptions

CategoriesManagerImpl:
This is the default implementation of CategoriesManager. It extends HibernateDaoSupport to execute HQL query to get the data.
Thread Safety:
This class is not technically thread-safe because it's mutable. However, it can be considered effectively thread-safe after configuration assuming that it's not changed after configuration and the passed in parameters are not changed by other threads at the same time because it holds no mutable state under such assumptions

com.topcoder.web.tc.impl.entity:
ResourceSubmission:
This class is simply the Java mapped class for table resource_submission, so that this table can be used in HQL.
Thread Safety:
This class is not thread-safe because it's mutable

ProjectInfo:
This class is simply the Java mapped class for table project_info, so that this table can be used in HQL.
Thread Safety:
This class is not thread-safe because it's mutable

ProjectGroupCategoryLookup:
This class is simply the Java mapped class for table project_group_category_lu, so that this table can be used in HQL.
Thread Safety:
This class is not thread-safe because it's mutable

Submission:
This class is simply the Java mapped class for table 'submission', so that this table can be used in HQL.
Thread Safety:
This class is not thread-safe because it's mutable

Project:
This class is simply the Java mapped class for table 'project', so that this table can be used in HQL.
Thread Safety:
This class is not thread-safe because it's mutable

ResourceInfo:
This class is simply the Java mapped class for table resource_info, so that this table can be used in HQL.
Thread Safety:
This class is not thread-safe because it's mutable

Resource:
This class is simply the Java mapped class for table 'resource', so that this table can be used in HQL.
Thread Safety:

This class is not thread-safe because it's mutable

LookupEntity:

This class simply has a name property for the subclasses to use. It represents a lookup entity.
Thread Safety:
This class is not thread-safe because it's mutable

ProjectCatalogLookup:

This class is simply the Java mapped class for table project_catalog_lu, so that this table can be used in HQL.
Thread Safety:
This class is not thread-safe because it's mutable

ProjectResult:

This class is simply the Java mapped class for table project_result, so that this table can be used in HQL.
Thread Safety:
This class is not thread-safe because it's mutable

ProjectPhase:

This class is simply the Java mapped class for table project_phase, so that this table can be used in HQL.
Thread Safety:
This class is not thread-safe because it's mutable

ProjectCategoryLookup:

This class is simply the Java mapped class for table project_category_lu, so that this table can be used in HQL.
Thread Safety:
This class is not thread-safe because it's mutable

Upload:

This class is simply the Java mapped class for table ‘upload’, so that this table can be used in HQL.
Thread Safety:
This class is not thread-safe because it's mutable

ContestEligibility:

This class is simply the Java mapped class for table ‘contest_eligibility’, so that this table can be used in HQL.
Thread Safety:
This class is not thread-safe because it's mutable

1.5 Component Exception Definitions

1.5.1 Custom exceptions

ContestServicesActionException:

This is thrown by the Struts actions of this component. It's thrown by BaseJSONParameterAction, ActiveContestsManagerAction, ContestStatusManagerAction, PastContestsManagerAction, CategoriesManagerAction.

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

CategoriesManagerException:

This is thrown if any error occurs in CategoriesManager. It's thrown by CategoriesManager(and its implementations).

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

ActiveContestsManagerException:
This is thrown if any error occurs in ActiveContestsManager. It's thrown by ActiveContestsManager(and its implementations).

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

ContestServicesConfigurationException:
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.

ContestStatusManagerException:
This is thrown if any error occurs in ContestStatusManager. It's thrown by ContestStatusManager(and its implementations).

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

PastContestsManagerException:
This is thrown if any error occurs in PastContestsManager. It's thrown by PastContestsManager(and its implementations).

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 and interceptors will be created by the Struts framework to process each user request.

When this component is used programmatically, the component is thread-safe to use because ActiveContestsManager, CategoriesManager, ContestStatusManager, PastContestsManager require implementations to be thread-safe and their implementations are thread-safe after configuration as stated in 1.4.

2. Environment Requirements

2.1 Environment

Development language: Java1.6 J2EE 1.5

Compile target: Java1.6 J2EE1.5

2.2 TopCoder Software Components

Base Exception 2.0 – used as the base for all custom exception classes.

JSON Object 1.0 – used to convert JSON string into JSON object

2.3 Third Party Components

Spring 3.0 (http://www.springsource.org/download)

Struts 2.2.3 (http://struts.apache.org/download.cgi)

Apache Log4j 1.2.15 (http://logging.apache.org/log4j/1.2/index.html)

Hibernate 3.2

3. Installation and Configuration

3.1 Configuration Parameters

Configuration for ActiveContestsManagerAction:

Name	Description	Value
activeContestsManager	The ActiveContestsManager for getting active contests.	com.topcoder.web.tc.ActiveContestsManager instance. It cannot be null. Required.

Please see the configuration of its base class BaseJSONParameterAction for more configuration parameters.

Configuration for BaseJSONParameterAction:

Name	Description	Value
parameterKey	The name of the http parameter that is a JSON string as the input to this action.	java.lang.String instance. It cannot be null or empty. Optional. Default to "parameter"
dateFormatString	The date format string used to parse dates in the JSON string.	java.lang.String instance. It cannot be null or empty. Optional. Default to "yyyy/MM/dd"

Configuration for CategoriesManagerAction:

Name	Description	Value
categoriesManager	The CategoriesManager for getting contest types, sub-types, and catalogs.	com.topcoder.web.tc.CategoriesManager instance. It cannot be null. Required.

Configuration for ContestStatusManagerAction:

Name	Description	Value
contestStatusManager	The ContestStatusManager for getting contest statuses.	com.topcoder.web.tc.ContestStatusManager instance. It cannot be null. Required.

Please see the configuration of its base class BaseJSONParameterAction for more configuration parameters.

Configuration for PastContestsManagerAction:

Name	Description	Value
pastContestsManager	The PastContestsManager for getting past contests.	com.topcoder.web.tc.PastContestsManager instance. It cannot be null. Required.

Please see the configuration of its base class BaseJSONParameterAction for more configuration parameters.

Configuration for ActiveContestsManagerImpl:

Name	Description	Value
projectNameInfoId	The id of the project_info_type_lu row that has name 'Project Name'.	long. It must be non-negative. Required.
registrationPhaseTypeId	The id of the phase_type_lu	long. It must be non-negative.

Name	Description	Value
	row that has name 'Registration'.	Required.
submissionPhaseTypeId	The id of the phase_type_lu row that has name 'Submission'.	long. It must be non-negative. Required.
firstPlaceCostInfoId	The id of the project_info_type_lu row that has name 'First Place Cost'.	long. It must be non-negative. Required.
paymentsInfoId	The id of the project_info_type_lu row that has name 'Payments'.	long. It must be non-negative. Required.
digitalRunFlagInfoId	The id of the project_info_type_lu row that has name 'Digital Run Flag'.	long. It must be non-negative. Required.
reliabilityBonusCostInfoId	The id of the project_info_type_lu row that has name 'Reliability Bonus Cost'.	long. It must be non-negative. Required.
digitalRunPointInfoId	The id of the project_info_type_lu row that has name 'DR points'.	long. It must be non-negative. Required.
activeStatusId	The id of the project_status_lu row that has name 'Active'.	long. It must be non-negative. Required.
finalReviewPhaseTypeId	The id of the phase_type_lu row that has name 'Final Review'.	long. It must be non-negative. Required.
openPhaseStatusId	The id of the phase_status_lu row that has name 'Open'.	long. It must be non-negative. Required.
submitterRoleId	The id of the resource_role_lu row that has name 'Submitter'.	long. It must be non-negative. Required.
contestSubmissionTypeId	The id of the submission_type_lu row that has name 'Contest Submission'.	long. It must be non-negative. Required.
activeSubmissionStatusId	The id of the submission_status_lu row that has name 'Active'.	long. It must be non-negative. Required.
failedScreeningSubmissionStatusId	The id of the submission_status_lu row that has name 'Failed Screening'.	long. It must be non-negative. Required.
failedReviewSubmissionStatusId	The id of the submission_status_lu row that has name 'Failed Review'.	long. It must be non-negative. Required.
completedWithoutWinSubmissionStatusId	The id of the submission_status_lu row that has name 'Completed Without Win'.	long. It must be non-negative. Required.

Configuration for ContestStatusManagerImpl:

Name	Description	Value
submissionPhaseTypeId	The id of the phase_type_lu row that has name 'Submission'.	long. It must be non-negative. Required.
finalReviewPhaseTypeId	The id of the phase_type_lu	long. It must be non-negative.

Name	Description	Value
	row that has name 'Final Review'.	Required.
openPhaseStatusId	The id of the phase_status_lu row that has name 'Open'.	long. It must be non-negative. Required.
winnerIdInfoId	The id of the project_info_type_lu row that has name 'Winner External Reference ID'.	long. It must be non-negative. Required.
externalReferenceIdInfoId	The id of the resource_info_type_lu row that has name 'External Reference ID'.	long. It must be non-negative. Required.
handleInfoId	The id of the resource_info_type_lu row that has name 'Handle'.	long. It must be non-negative. Required.
secondPlaceIdInfoId	The id of the project_info_type_lu row that has name 'Runner-up External Reference ID'.	long. It must be non-negative. Required.
projectNameInfoId	The id of the project_info_type_lu row that has name 'Project Name'.	long. It must be non-negative. Required.
registrationPhaseTypeId	The id of the phase_type_lu row that has name 'Registration'.	long. It must be non-negative. Required.
firstPlaceCostInfoId	The id of the project_info_type_lu row that has name 'First Place Cost'.	long. It must be non-negative. Required.

Configuration for PastContestsManagerImpl:

Name	Description	Value
projectNameInfoId	The id of the project_info_type_lu row that has name 'Project Name'.	long. It must be non-negative. Required.
completedStatusId	The id of the phase_status_lu row that has name 'Completed'.	long. It must be non-negative. Required.
approvalPhaseTypeId	The id of the phase_type_lu row that has name 'Approval'.	long. It must be non-negative. Required.
winnerIdInfoId	The id of the project_info_type_lu row that has name 'Winner External Reference ID'.	long. It must be non-negative. Required.
externalReferenceIdInfoId	The id of the resource_info_type_lu row that has name 'External Reference ID'.	long. It must be non-negative. Required.
handleInfoId	The id of the resource_info_type_lu row that has name 'Handle'.	long. It must be non-negative. Required.
submitterRoleId	The id of the resource_role_lu row that has name 'Submitter'.	long. It must be non-negative. Required.
contestSubmissionTypeId	The id of the submission_type_lu row that	long. It must be non-negative. Required.

Name	Description	Value
	has name 'Contest Submission'.	
activeSubmissionStatusId	The id of the submission_status_lu row that has name 'Active'.	long. It must be non-negative. Required.
failedScreeningSubmissionStatusId	The id of the submission_status_lu row that has name 'Failed Screening'.	long. It must be non-negative. Required.
failedReviewSubmissionStatusId	The id of the submission_status_lu row that has name 'Failed Review'.	long. It must be non-negative. Required.
completedWithoutWinSubmissionStatusId	The id of the submission_status_lu row that has name 'Completed Without Win'.	long. It must be non-negative. Required.
winnerProfileLinkTemplate	The winner profile link template. This is a URL that points to the profile page of the winner, which must have a variable "%id%". This variable will be replaced with the actual winner id.	java.lang.String instance. It cannot be null or empty. Required.
passingScreeningScore	The minimal screening score that passes screening.	double. It must be non-negative. Required.
registrationPhaseTypeId	The id of the phase_type_lu row that has name 'Registration'.	long. It must be non-negative. Required.
submissionPhaseTypeId	The id of the phase_type_lu row that has name 'Submission'.	long. It must be non-negative. Required.

The above tables' configuration should be done in Spring XML. The attached "applicationContext.xml" is a sample Spring file.

3.2 Dependencies Configuration

Log4j should be configured properly.

3.3 Package Structure

com.topcoder.web.tc
com.topcoder.web.tc.action
com.topcoder.web.tc.dto
com.topcoder.web.tc.impl
com.topcoder.web.tc.impl.entity

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

Please see the demo.

4.3 Demo

A sample Spring configuration file “applicationContext.xml” is attached for reference only.

4.3.1 API Usage

```
        // Load the Spring context file
        ClassPathXmlApplicationContext applicationContext = new ClassPathXmlApplicationContext(
            "applicationContext.xml");

        // Get the ActiveContestsManager from Spring
        ActiveContestsManager manager = (ActiveContestsManager) applicationContext
                        .getBean("activeContestsManagerBean");

        // Create a new ActiveContestFilter
        ActiveContestFilter filter = new ActiveContestFilter();

        // The filtered contests' name should start with "TC Refactoring"
        filter.setContestName("TC Refactoring");
        // The lower bound of the prize is -infinite
        filter.setPrizeLowerBound(Filterable.OPEN_INTERVAL);
        // The upper bound of the prize is 1000
        filter.setPrizeUpperBound(1000);

        // This call retrieves all active contests whose name starts with "TC Refactoring" and prize
<= 1000
        // sorted in ascending order by "contestName" property. The page size is 10 and only the 2nd
page
        // should be returned.
        List<ActiveContestDTO> activeContests =
manager.retrieveActiveContests("projectGroupCategory.name",
            SortingOrder.ASCENDING, 2, 10, filter);

        // Get the CategoriesManager from Spring
        CategoriesManager categoriesManager = (CategoriesManager) applicationContext
                        .getBean("categoriesManagerBean");

        // Get all catalogs
        List<String> catalogs = categoriesManager.getCatalogs();

        // Get all types
        List<String> types = categoriesManager.getContestTypes();

        // Get all types of catalog 'application'
        types = categoriesManager.getContestTypes("application");

        // Get all sub-types of type 'architecture'
               List<String> subTypes = categoriesManager.getContestSubTypes("architecture");
```

The usage of ContestStatusManager and PastContestsManager are similar to ActiveContestsManager.

4.3.2 Struts Usage

Since Struts2 does all the work there is nothing to show in demo in fact. Here an example of getting active contests through Struts.

Suppose that a page is used to get active contests. It should assemble the input parameters into a JSON string, such as:

```
{ columnName:”contestName”,
  sortingOrder:”ASCENDING”,
  pageNumber: “2”,
  pageSize: “10”,
  filter: {contestName:”TC Refactoring%”, prizeLowerBound:”-1”, prizeUpperBound:”1000”}
}
```

Then launch a HTTP request to the URL that is handled by the action ActiveContestsManagerAction. This request should have a parameter whose name is “parameter” and value is the above JSON string.

Then the ActiveContestsManagerAction will retrieve all active contests whose name starts with "TC Refactoring" and prize <= 1000 sorted in ascending order by "contestName" property. The page size is 10 and only the 2nd page should be returned.

The retrieved List<ActiveContestDTO> will be put into request scope with name “activeContests”, so that JSP can access it.

5. Future Enhancements

None