

Cockpit Phase Management Persistence 1.0 Component Specification

1. Design

This component provides an implementation of the PhaseManager interface from the Phase Management component.

The implementation adapt the ContestManager interface from the Studio Contest Manager component to provide the necessary functionality.

This component allows the Phase Management component to work with studio contests.

The design provides only the manager class and the custom exceptions. The main class implements only the necessary operations, while the other operations do nothing. The attributes of the class are configurable. The cache is used to cache the contest statuses.

1.1 Design Patterns

1.1.1 Strategy

The manager can use strategically the contest manager and cache implementations.

1.1.2 Adapter

The manager adapts the contest manager interface. It maps the Contest, ContestStatus to Project, Phase, PhaseType.

1.2 Industry Standards

JNDI

EJB

1.3 Required Algorithms

1.3.1 Logging

The logging is required only in the *start* and *end* method. Log the business logic with the necessary information: the logic of the operation, the phase id (contest status id), the phase type name (contest status name), the project id (contest id). Any additional information is at the developer's discretion. Log using the *INFO* level.

The logging in the other methods is at the developer's discretion: you can log the business logic also for the other methods.

The logging of exceptions, method entrance or exit, or other levels of logging is at developer's discretion.

1.3.2 Entity mapping

This component is an adapter from the ContestManager to the PhaseManager,

therefore we should map the Contest entities to the Phase entities. Only the Contest entities --> Phase Entities way is necessary

Contest --> Project

The Contest entity is mapped to the Project Entity. This is the table of mapping:

Contest	Project
contestId	id

The other fields of the Contest must be added to Project as attribute. The key of the attribute will be the name of the field in the Contest entity as String plus the prefix "contest", the value of the attribute will be the value of the field. For example forumId will be added to the Project as: "contestForumId" --> forumId attribute ($a \rightarrow b$ where a is the name of attribute and b is the value of attribute). The camelCase is used.

The collection are added directly except for the config attribute.

The ContestConfig instances must be added with:

contestConfig.property.propertyId --> contestConfig.value.

For the contest.status property see the following section, it must be mapped to a collection of Phase entities.

Add also the PhaseStatus to the Phase using the related *PhaseStatus* section.

When create the new Project, create an empty WorkDays with the default WorkDaysFactory.

ContestStatus --> Phase

The ContestStatus is mapped to the Phase entity. This is the table of mapping:

ContestStatus	Phase
ContestStatus.contestStatusId	Phase.id
ContestStatus.name	Phase.PhaseType.name
ContestStatus.contestStatusId	Phase.PhaseType.id

The other fields of the ContestStatus (description,name) must be added to Phase as attribute. The key of the attribute will be the name of the field in the Contest entity as String, the value of the attribute will be the value of the field, with the same mechanism described for the Contest entity.

Add the contestStatus.statuses item to the attributes using: status.name -->

ContestStatus (directly the ContestStatus without conversion)

In the case of Contest.status, the status is mapped to many phases entities in the Project entity, these phases are defined by PhaseType.name.

Firstly get all contest statuses from the cache or from the ContestManager (if the cache is not null and is empty then fill the cache with the statuses, lazy mode). At this point match the name of the contest statuses retrieved using the configurable names of the phases provided as fields in the cockpit phase manager class (the *PhaseTypeName* attributes). Use the following table to create the related Phase entities in the Project.phases collection:

Contest.status.name	Phase to create and add to the Project:
----------------------------	--

	PhaseType.name
Draft	Draft Scheduled
Scheduled	Scheduled Active
Active	Active Action Required Insufficient Submissions - ReRun Possible
Action Required	Action Required Completed In Danger
In Danger	In Danger Completed Abandoned
Insufficient Submissions - ReRun Possible	Insufficient Submissions - ReRun Possible Extended Abandoned
Extended	Extended Action Required Insufficient Submissions
Repost	Repost Action Required Insufficient Submissions - ReRun Possible
Insufficient Submissions	Insufficient Submissions Cancelled Abandoned
No Winner Chosen	No Winner Chosen Cancelled Abandoned Repost

Example:

- 1) You have the Contest.status.name="Extended".
- 2) Create 3 Phase entities
- 3) Retrieve the contest statuses from the all contest statuses equal to the 3 names : "Extended", "Action Required " and "Insufficient Submissions" (note that for each status the current status, in this case "Extended", is always added). Use always the configurable attributes of the cockpit phase manager class.
- 4) set the id of Phase using the table of ContestStatus --> Phase mapping
- 5) create the 3 PhaseType and use the table of ContestStatus --> Phase mapping to map the 3 ContestStatus to the 3 PhaseType
- 6) set the PhaseTypes to the Phase entities
- 7) add the Phase entities to the Project

When create the new Phase, specify the length will be the

contest.endDate-contest.startDate in milliseconds.

PhaseStatus

Use the PhaseStatusEnum.OPEN to map the phase status, when you get the contest status this means that the contest status is open, the contest is in that status.

1.3.3 Update the contest to the new status

This algorithm is used in the *start* and *end* methods. This is the algorithm:

- get the Project from the Phase
- get the Project.id : this will be used as contestId
- map the Phase.PhaseType to the ContestStatus using the algorithm described in the previous section (get all contest statuses and check the name)
- from the mapped ContestStatus get the contestStatusId
- call the ContestManager for updating the contest status passing the contestId and the contestStatusId

1.3.4 Get the phase handler

This algorithm is used in the canStart/start/canEnd/end methods. The documentation of methods described the values used by this algorithm. This is the algorithm:

- create the HandlerRegistryInfo with the PhaseType of Phase and PhaseOperationEnum described in the documentation of method
- get the PhaseHandler from the handlers map using the HandlerRegistryInfo key

1.4 Component Class Overview

1.4.1 *com.topcoder.management.persistence.clientcockpit*

CockpitPhaseManager

This is main (and only) class of this component. It is an adapter of the PhaseManagerInterface to the part of ContestManager interface related to ContestStatus. This manager will be used in the Cockpit application. The methods of this class map the Project and Phase instance to Contest and ContestStatus. Some methods do nothing, there is not a mapping of these methods for this cockpit manager implementation. This class uses an instance of contest manager to delegate the work. The handlers will be the handlers from the Cockpit Phase handlers component. They are used to perform the methods related to the phases: start/canStart and end/canEnd methods. A cache is used to cache the contest statuses: they change rarely, anyway it is optional. The logging is optional. Refer to the Algorithm Section in CS for the mapping of Contest to Project and ContestStatus to Phase,PhaseType,PhaseStatus. The class provides also the phase types name configurable: these fields have the default value defined (it's the "initial value").

The class provides an empty constructor and the setters and getters for the

instance.

This will be useful for a setter injection of the instances used by this class.

1.5 Component Exception Definitions

CockpitPhaseManagementException

Thrown by various CockpitPhaseManager methods when an error occurs. If the error was the result of an internal exception (such as a persistence problem), the CockpitPhaseManagementException will have an associated wrapped exception. It is used to wrap the exception thrown by the ContestManager and the exceptions thrown by the PhaseHandlers.

CockpitConfigurationException

This exception is thrown when some errors occur in the configuration: properties are missing or in the bad format.

1.6 Thread Safety

The component consists only in a single class, therefore the thread safety discussion is the same of the thread safety discussion of the class.

The setters and getters of the class are not thread safe. I suppose that they are used in a single thread, when the manager is constructed with the setters injection. The register/unregister methods will be used in a single thread too. The thread safety of the other methods depends from the thread safety of the ContestManager and the cache interface. Supposing that they are thread safe (in their documentation they are required to be thread safe) then these methods are thread safe. Considering that only these methods will be used in a possible multiple threads environment, the class under these conditions can be considered thread safe and therefore the component can be considered thread safe.

2. Environment Requirements

2.1 Environment

- Development language: Java 1.5
- Compile target: Java 1.5, 1.6

2.2 TopCoder Software Components

- Configuration Manager 2.1.5
 - Used for configuration of the manager
- Object Factory 2.0.1
 - Used to create Cache, HandlerRegistryInfo and PhaseHandlers in the manager
- Logging Wrapper 2.0

- Used for logging the operations.
- Phase Management 1.0.4
 - It contains the PhaseManager implemented by the manager of this component.
- Project Phases 2.0
 - Used for the Project and Phase entities
- Studio Contest Manager 1.0
 - It contains the ContestManager interface used by the manager
- Contest and Submission Entities 1.0
 - It contains the Contest and Submission entities used by the manager
- JNDI Context Utility 2.0
 - Used for the JNDI operations
- Simple Cache 2.0.2
 - Used for cache the contest statuses
- Workdays 1.0.1
 - Project Phases depends on it

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

There are no third party components that need to be used directly by this component.

3. Installation and Configuration

3.1 Package Names

`com.topcoder.management.persistence.clientcockpit`

3.2 Configuration Parameters

3.2.1 CockpitPhaseManager

Parameter	Description	Details
objectFactoryNamespace	Namespace to use with the	String not null and not empty

	ConfigManagerSpecificationFactory Required	
jndiUtilNamespace	Namespace to use with the JNDIUtil Optional, in this case the empty construct is used. It is constructed only if the contestManagerName property is present.	String not null and not empty
contestManagerName	The name of the ContestManager bean, used by JNDIUtil Optional, in this case ObjectFactory is used	String not null and not empty
contestManagerKey	The key to construct the ContestManager with ObjectFactory Required only if the contestManagerName is not set	String not null and not empty
logName	Name of the Log Optional, the logging is optional	String not null and not empty
cacheContestStatusesKey	The key to construct the Cache implementation for the contest statuses caching Optional, the caching is optional	String not null and not empty
handlers	This property contains several properties which define the handlers mapping Required	A property with multiple properties
handlers.property key	The keys of the properties of handlers are the strings to construct the HandlerRegistryInfo with Object Factory. The string is in CSV format: "phaseTypeId,phaseTypeName,phaseOperationEnumName" Required	String not null and not empty and in CSV value. An example: "2,actionRequired,start"
handlers.property value	The values of the properties of handlers are the keys to construct the PhaseHandler with Object Factory Required	String not null and not empty
draftPhaseTypeName	The name for the Draft PhaseType Optional, the default value is defined in the related attribute	String not null and not empty
scheduledPhaseTypeNam	The name for the Scheduled	String not null and not empty

e	PhaseType Optional, the default value is defined in the related attribute	
activePhaseTypeName	The name for the Active PhaseType Optional, the default value is defined in the related attribute	String not null and not empty
inDangerPhaseTypeName	The name for the In Danger PhaseType Optional, the default value is defined in the related attribute	String not null and not empty
insufficientSubmissionsReRunPossiblePhaseName	The name for the Insufficient Submissions ReRun Possible PhaseType Optional, the default value is defined in the related attribute	String not null and not empty
extendedPhaseName	The name for the Extended PhaseType Optional, the default value is defined in the related attribute	String not null and not empty
repostPhaseName	The name for the Repost PhaseType Optional, the default value is defined in the related attribute	String not null and not empty
insufficientSubmissionsPhaseName	The name for the Insufficient Submissions PhaseType Optional, the default value is defined in the related attribute	String not null and not empty
noWinnerChosenPhaseName	The name for the No Winner PhaseType Optional, the default value is defined in the related attribute	String not null and not empty
completedPhaseName	The name for the Completed PhaseType Optional, the default value is defined in the related attribute	String not null and not empty
abandonedPhaseName	The name for the Abandoned PhaseType Optional, the default value is defined in the related attribute	String not null and not empty
cancelledPhaseName	The name for the Cancelled PhaseType Optional, the default value is defined in the related attribute	String not null and not empty

3.3 Dependencies Configuration

3.3.1 TopCoder dependencies

All the dependencies are to be configured according to their component specifications.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow the instructions.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

None

4.3 Demo

4.3.1 Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<Config name="CockpitPhaseManager">
  <Property name="objectFactoryNamespace">
    <Value>objectFactory</Value>
  </Property>

  <Property name="jndiUtilNamespace">
    <Value>jndi</Value>
  </Property>

  <Property name="contestManagerName">
    <Value>contestManagerRemote</Value>
  </Property>

  <Property name="logName">
    <Value>myLog</Value>
  </Property>

  <Property name="cacheContestStatusesKey">
    <Value>myCache</Value>
  </Property>

  <!-- Defien the phase handlers -->
  <Property name="handlers">
    <Property name="1,draft,start">
      <Value>draftPhaseHandler</Value>
    </Property>
    <!-- It's the same handler also for end -->
    <Property name="1,draft,end">
      <Value>draftPhaseHandler</Value>
    </Property>
  </Property>
</Config>
```

```

        <Property name="2,scheduled,start">
            <Value>scheduledPhaseHandler</Value>
        </Property>
        <!-- ...and so on, define all the cockpit phase handlers
-->
    </Property>

</Config>

```

4.3.2 Usage

```

    // create the phase manager with the namespace defined in the setup
    PhaseManager manager = new CockpitPhaseManager
        ("CockpitPhaseManager");

    // Suppose that a Contest exists in the persistence layer
    // This contest has the name "Kyx Persistence", the id=17 and the
    // status is "Draft"
    // This contest has the two documents: the first with the id=25
    // and the second with
    // the id=30

    // Get the phases, get the contest data
    Project project = manager.getPhases(17);
    // the data of the contest is in the Project entity

    // get the phases
    Phase[] phases = project.getAllPhases();
    // the phases contains two items: "Draft" and "Scheduled" phases
    // these phases have the ids equal to the ContestStatus "Draft"
    // and "Scheduled"

    // get the attributes
    Map attributes = project.getAttributes();
    // these attributes contains the data from the Contest
    // there are two attributes (in addition to the other attributes)
    // related to the
    // documents
    // these attributes are: "document 25" --> the Document,
    // "document 30" --> the Document

    // get all phase types
    PhaseType[] allPhaseTypes = manager.getAllPhaseTypes();
    // the phase types are retrieved
    // these phase types contain the information of all Contest
    // Statuses
    // the id and the names are equal to the Contest Statuses, they
    // will be "Draft",
    // "Action Required" etc..

    // get the all phase status
    PhaseStatus[] phaseStatus = manager.getAllPhaseStatuses();
    // the statuses are the same, retrieved from the values of the
    // PhaseStatus class

```

```

// Suppose that the scheduled phase is the 1 item
Phase scheduledPhase = phases[1];

// start the scheduled phase to the contest
// this scheduled phase has the PhaseType.name equal to
  "Scheduled"
// the "scheduled" prefix in the phase status IS DIFFERENT from
  the  "Scheduled"
// Phase Type: this means only that the phase can be started
// the phase status has only 3 values: SCHEDULED, OPEN and CLOSE
  (these states will
// be read by the cockpit phase handlers)
// set the phase status
scheduledPhase.setPhaseStatus(PhaseStatus.SCHEDULED);

// start the "Scheduled" phase, the operator will be ignored by
  the phase handlers
manager.start(scheduledPhase, null);
// the handler with the related
// HandlerRegistryInfo(scheduledPhase,PhaseOperation.START) will
  be called
// now the "Scheduled" phase is started
// now the ContestStatus has the "Scheduled" contest status

// Now I want to end the "Scheduled" phase
// I set the status to OPEN
scheduledPhase.setPhaseStatus(PhaseStatus.OPEN);
// (probably this operation is not necessary because the "start"
  method should
// change the status, I wait the response from the architect in
  the  forum, this will
// be fixed in the final fix phase).

// end the "Scheduled" phase, the operator will be ignored by the
  phase handlers
manager.end(scheduledPhase, null);
// the handler with the related
// HandlerRegistryInfo(scheduledPhase,PhaseOperation.END) will be
  called
// now the "Scheduled" phase is ended
// now the ContestStatus has already "Scheduled" contest status

```

5. Future Enhancements

None.