Java Custom Cockpit Phase Management Persistence

1. Scope

1.1 Overview

This component provides an implementation of the PhaseManager interface from the Phase Management component. The implementation will adapt the ContestManagerBean from the Studio Contest Manager component to provide the necessary functionality. This component will allow the Phase Management component to work with studio contests.

1.1.1 Version

1.0

1.2 Logic Requirements

1.2.1 Persistence Implementation

The persistence implementation will utilize the ContestManager interface, from the Studio Contest Manager component, which will be deployed in an EJB 3.0 container. The constructors for the persistence implementation should expect either a ContestManagerRemote bean being directly provided, or a name that can be used with JNDI to retrieve the bean.

1.2.2 Phase Handlers

The implementation should contain a mapping of HandlerRegistryInfo instances to PhaseHandler implementations, which will be loaded through configuration. The PhaseHandlers will come from the Cockpit Phase Handlers component.

1.2.3 Adapter pattern

This component will adapt the entities from the "Contest and Submission Entities" component to work with the entities from Project

Phases. The mapping should look something like this:

Project	**Contest**
Project.id	Contest.contestId

The Contest entity should also be added to the Project.attributes collection

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

The ContestStatus entity should also be added to the Phase.attributes collection

1.2.4 Getting phases for a project

The phases to add to the "phases" collection for a Project depend on the status of the particular contest being used to create the Project. To get the contest statuses, use ContestManager.getAllContestStatuses, and find those whose name matches the names described in the table below, converting it to a Phase entity and adding the filled Phase entity to Project.phases. Make sure to also add a Phase entity for the current contest status as well.

Current contest status	**ContestStatus / Phases to create and add to the Project**
Draft	Scheduled
Scheduled	Active
Active	Action Required Insufficient Submissions - ReRun Possible
Action Required	Completed

	In Danger
In Danger	Completed Abandoned
Insufficient Submissions - ReRun Possible	Extended Abandoned
Extended	Action Required Insufficient Submissions
RePost	Action Required Insufficient Submissions - ReRun Possible
Insufficient Submissions	Cancelled Abandoned
No Winner Chosen	Cancelled Abandoned Repost

1.2.5 Implementation details

The following details can be used when implementing the manager interface.
updatePhases: Do nothing
getPhases: Use ContestManager.getContest and ContestManager.getContestStatus

getAllPhaseTypes: Use ContestManager.getAllContestStatuses

getAllPhaseStatuses: Use ContestManager.getAllContestStatuses

getPhase: Use ContestManager.getContestStatus

getPhases: Use ContestManager.getContestStatus
canStart: Get the phase handler for the Phase.PhaseType.Name value, and the PhaseOperationEnum.START value. Then, call canPerform on the phase handler retrieved

start: Update the contest to the new status. Get the phase handler for the Phase.PhaseType.Name value, and the PhaseOperationEnum.START value. Then, call perform on the phase handler retrieved
canEnd: Get the phase handler for the Phase.PhaseType.Name value, and the PhaseOperationEnum.END value. Then, call canPerform on the phase handler retrieved.

end: Update the contest to the new status. Get the phase handler for the Phase.PhaseType.Name value, and the PhaseOperationEnum.END value. Then, call perform on the phase handler retrieved
canCancel: Return "false"

cancel: Do nothing

1.2.6 Logging

All phase operations should be logged at designer chosen levels, making it easy to debug and track operations performed by the component.

1.3 Required Algorithms

The adaptation algorithm used is integral to this component, and should be described in detail to ensure that it is implemented correctly.

1.4 Example of the Software Usage

The Auto Pilot component will be used to handle phases of contests in the Client Cockpit application. It uses the Phase Management component, and this component will provide the bridge between Phase Management and the Client Cockpit.

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
- Compile target: Java1.5 and Java1.6

2.1.4 Package Structure

com.topcoder.management.phase.clientcockpit

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

- The connection to the ContestManager

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

JNDI

3.2.2 TopCoder Software Component Dependencies:

- Studio Contest Manager 1.0
- Contest and Submission Entities 1.0
- Phase Management 1.0
- Logging Wrapper 2.0

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

3.2.3 Third Party Component, Library, or Product Dependencies:

None.

3.2.4 QA Environment:

- Solaris 7

- RedHat Linux 7.1
- Windows 2000
- Windows 2003
- Informix 10.0

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

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 Poseidon.