Online Review Phases 1.0 Component Specification

The Online Review application defines a set of phase types. This component provides the plug-ins to the Phase Management component, whose logic is to check if these phases can be executed. Extra logic to execute the phases is also provided.

The component provides a set of phase handler classes. They implement PhaseHandler interface to provide plug-in handlers for Phase Management component. Phase Management can load these handler classes from a configuration files. All handlers has constructor that can load settings from a configuration namespace.

Each handler provides two methods, that are canPerform () and perform(). canPerform() method determine if a phase can start or stop. While perform() execute some additional logic to start or stop a phase. These methods examine the input phase status to choose the action. If phase status is “Scheduled”, they will check for can start or execute additional start logic. If phase status is “Open”, they will check for can stop or execute additional stop logic.

The phase handlers in this component provide the logic to handle the following phases:

- Registration
- Submission
- Screening
- Review
- Appeals
- Appeals Response
- Aggregation
- Aggregation Review
- Final Fix
- Final Review
- Approval

The phase handlers can send email to a group of users associated to the timeline notification of the current project. User can configure to send email when start phase or when end phase, in both start or end or no email sending at all.

This component provides some extra useful features such as:

- Adding some utility classes for getting value from lookup tables. They are phase status, phase type, resource role and submission status. These lookup tables are frequently used in phase related operations.
- Provide caching mechanism for lookup id. Because lookup id and lookup value pairs are not changed per database installation, caching them will minimize the database queries.

1.1 Design Patterns

Strategy pattern is used in phase handler classes. They implement the PhaseHandler interface to provide plug-in handlers for the Phase Management component.

1.2 Industry Standards

JDBC, XML, SQL

1.3 Required Algorithms

1.3.1 Loading configuration property [Common]

This is the logic of AbstractPhaseHandler constructor
Get the value of the 'ConnectionFactoryNS' property in the give namespace – required
Create a new instance of DBConnectionFactoryImpl using the above value and assign to the 'factory' field.
Get the value of the 'ManagerHelperNamespace' property in the give namespace - optional
    If the property does not exist, create the ManagerHelper field using default constructor
    Else, create the ManagerHelper field using the configured namespace.
Get the value of the 'ConnectionName' property - optional. Assign its value to 'connectionName' field if it exists.
If 'StartPhaseEmail' property exist, set 'sendStartPhaseEmail' field to true. Then load sub properties for 'StartPhaseEmail'
If 'EndPhaseEmail' property exist, set 'sendEndPhaseEmail' field to true. Then load sub properties for 'EndPhaseEmail'

Loading email sub properties (xx can be start of end):
    Get the value of the 'EmailTemplateSource' property - required. Assign its value to 'xxTemplateSource' field.
    Get the value of the 'EmailTemplateName' property - required. Assign its value to 'xxTemplateName' field.
    Get the value of the 'EmailSubject' property - required. Assign its value to 'xxEmailSubject' field.
    Get the value of the 'EmailFromAddress' property - required. Assign its value to 'xxEmailFromAddress' field.

Throw ConfigurationException if any required field is missing or empty.

1.3.2 Common logic for accessing database methods [Common]

To satisfy the requirement of connection cannot be cached. Methods that access database must create connection at the beginning and close connection when finished.

With methods that update database, transaction logic should be provided as below to ensure data consistence.
Create connection using the 'factory' instance and the configured DB connection name (connectionName).
If connectionName is null, create the default connection.

Set the connection’s auto commit to false.
Do create/update.
If error occurred, call rollback, and then throw PhaseHandlingException that wrap the error.
Call commit if everything is fine.
Close the connection in finally block.

1.3.3 Email sending when changing phase [Common]

This method is used in every phase handler class, perform() method to send email to the related users.

First check value of 'sendStartPhaseEmail' and 'sendEndPhaseEmail'.
    If phase status is 'Scheduled' and sendStartPhaseEmail=true --> Initialize and send start email
    If phase status is 'Open' and sendEndPhaseEmail=true --> Initialize and send end email
    Otherwise, do nothing

Get email addresses to send:
    Use Resource Management to get all notification ids for a project. The notification ids here are
    the external id of users that need to be notified when a phase change.

```
        Lookup project info type id for "Timeline Notification" use
ProjectInfoTypeLookupUtility class. (typeId)
        long[] externalIds =
ResourceManager.getNotifications(phase.getProject().getId(), typeId);

    Use User Project Data Store component to load users/project information
        ExternalUser[] users = UserRetrieval.retrieveUsers(externalIds);
        ExternalProject project =
ProjectRetrieval.retrieveProject(phase.getProject().getId());


Use DocumentGenerator
// instantiate the Document Generator
DocumentGenerator instance = DocumentGenerator.getInstance();

// get a template (xx can be start or end)
Template template = instance.getTemplate(xxTemplateSource, xxTemplateName);

// for each external user, set field values
TemplateFields root = instance.getFields(template);
Node[] nodes = root.getNodes();
for (int i = 0; i <  nodex.length; i++) {
if (nodes[i] instanceof Field) {
    Field field = nodes[i];
    // Set field value using field.setValue() method, base on field.getName()
}

String emailContent = instance.applyTemplate(root);

Send email
    TCSEmailMessage message = new TCSEmailMessage();
    message.setSubject(xxEmailSubject);
    message.setBody(emailContent);
    message.setFromAddress(xxFromAddress);
    message.setToAddress(ExternalUser.getEmail());
    EmailEngine.send(message);


The map between field name and value to set:
    PROJECT_NAME -> ExternalProject.getName()
    PROJECT_VERSION --> ExternalProject.getVersion()
    PHASE_TYPE --> Phase.getPhaseType().getName()
    PHASE_OPERATION --> "start" or "end" depends on the input phase status.
        If status is "Scheduled" -> "start"
        If status is "Open" -> "end"
    PHASE_TIMESTAMP --> The time of performing the phase
    USER_FIRST_NAME --> ExternalUser.getFirstName()
    USER_LAST_NAME --> ExternalUser.getLastName()
    USER_HANDLE --> ExternalUser. getHandler()

Those field names are constants. They are and used in a template of DocumentGenerator to
generate documents.
```

1.3.4 Using various related manager components [Common]

```
This component uses these components for searching/updating database:
Deliverable Management: UploadManager interface. Default implementation is
PersistenceUploadManager class.
Resource Management: ResourceManager interface. Default implementation is
PersistenceResourceManager class.
Review Management: ReviewManager interface. Default implementation is
InformixReviewManager class.
```

```
Scorecard Management: ScorecardManager interface. Default implementation is
InformixScorecardManager class.
Project Management: ProjectManager interface. Default implementation is
InformixProjectManager class.

Auto Screening Management: ScreeningManager interface. Default implementation is
DefaultDBScreeningManager class.

These default implementations of these interfaces require initialization of
DBConnectionFactory, SearchBundle and IDGenerator components. These components should be
properly configured. See their component specifications for configuration information.

All of those interface provide search() method for searching.
search() method receives a parameter of type Filter, a class of Search Builder
component.
Filter can be created by some utility methods provided in those management components.
Filter can be created for various search conditions and can be combined using
AND/OR/NOT.

Some phase handler's perform method need to update database.
In the update methods, operator parameter is required. Simply pass the "operator"
parameter to it.
```

1.3.5 Loading manager instance use reflection [Common]

ManagerHelper class loads the manager from the settings in a configuration namespace. The properties are detailed in section “3.2 Configuration Parameters”. Their values are used to create manager instances use reflection. The phase handler classes keep this class as a field to use when needed.
Following is details:

- For ScorecardManager, ReviewManager and ProjectManager:
  - If “Namespace” property does not present, default constructor will be used.
  - Else, use the namespace to pass to the constructor.
- For ResourceManager and UploadManager, the constructor parameters are SearchBundle and IDGenerator instances.
  - Create SearchBundleManager instance.
  - Use SearchBundleManager to create SearchBundle instances from the configured names.
  - Use IDGeneratorFactory to create IDGenerator instances from the configured names.
- For ScreeningManager:
  - If “Namespace” property does not present, use ScreeningManagerFactory.createScreeningManager() method to create the instance.
  - Else, use ScreeningManagerFactory.createScreeningManager(String) method to create the instance.

Object Factory component is recommended to load these instances using reflection.

1.3.6 Lookup values [Common]

```
Logic for lookUpId() methods in the utility classes:
- Look at 'cachedPairs' map to see if parameter 'value' exists as a key
    If yes
        Return the cached id
    Else:
        Query database for lookup id base the given value
        Save the pair to 'cachedPairs' map. Key is "lookup value", value is "lookup id"
        Return the id
    End If

Select commands (? will be the 'value' parameter)
    For SubmissionStatusLookupUtility
        SELECT submission_status_id FROM submission_status_lu WHERE name = ?
```

```
    For ResourceRoleLookupUtility
        SELECT resource_role_id FROM resource_role_lu WHERE name = ?
    For PhaseTypeLookupUtility
        SELECT phase_type_id FROM phase_type_lu WHERE name = ?
    For PhaseStatusLookupUtility
        SELECT phase_status_id FROM phase_status_lu WHERE name = ?
```

1.3.7 Check if a phase start time is reached [Routine]

```
Call phase.calcStartDate() method to get the date time when the given phase can start.
If current date time is later than or equal to phase start time, return true.
```

1.3.8 Check if a phase end time is reached [Routine]

```
Call phase.calcEndDate() method to get the date time when the given phase can end.
If current date time is later than or equal to phase end time, return true.
```

1.3.9 Check if all dependencies of a phase have stopped [Routine]:

```
Get an array of phase's dependencies
    dependencies[]=phase.getAllDependencies()

Return true if dependencies.length = 0

For each dependency
    Get its phase using getDependency() method (subPhase)
    Get phase status name using subPhase.getPhaseStatus().getName()
    If the phase status name is not "Closed", return false
End For

Return true (indicate that all dependency phases has status "Closed")
```

1.3.10 Locating phases [Routine]

```
In some methods, from the current phase we need to find a backward phase or a forward
phase. For example, from Aggregation phase, we may need to go back and find the nearest
Review phase. Or from Submission, we may need to go forward and find the nearest
Screening phase.

Get the current project
    currentPhase.getProject() (project)

Get all phases belong to the project. Note that the phases are sorted
    Using project.getAllPhases() (phases[])

Find the index of the currentPhase in phases[] (index)
    Using currentPhase.getId() and compare with instances in phases[]

To find a nearest backward phase of a type, start from index-1 and decreasing
To find a nearest forward phase of a type, start from index+1 and increasing

If cannot find the nearest backward/forward case of the given type, throw
PhaseHandlingException
```

1.3.11 Search resource base on resource role names and phase id [Routine]

```
Input:
    - resourceRoleNames(String[]): An array of resource role  name to search
    Resource is people who assigned for a phase. Resource role can be "Submitter",
"Screener", "Reviewer", "Aggregator", etc.
    - phaseId: The phase id to search for resource
Output:
    An array of Resource instance (resources[])
```

```
Lookup resource ids for the resource role names use "ResourceRoleLookupUtility" class.

Search using Resource Management
    Create filter with resource_role_id IN [roleIds] AND phase id = phase.getId()
    Resource[] resources = ResourceManager.search(filter)
```

1.3.12 Search all reviews for a phase base on resource roles [Routine]

```
Input:
    - phaseId(long): Id of the phase to search for reviews
    - resourceRoleName(String[]): The name of the reviewer. For screening review, the
name can be "Primary Screener" and "Screener". For review, it is "Reviewer". For
aggregation, it is "Aggregator"
Output:
    - reviews[]: An array of reviews of some type (screening review scorecard, main
review  scorecard or aggragation review scorecard)

Search the reviewIds using Resource Management:
    Search for resources with resource_role_name IN resourceRoleName[] AND
phase_id=phaseId
    Resource[] reviewers = ResourceManager.search(filter)
    Create an array of reviewerIds from reviewers[] array using reviewer.getId()
    Use Review Management to search for review with reviewer id IN [reviewerIds array]
    Review[] reviews = ReviewManager.search(filter)
```

1.3.13 Check screening type [Routine]

```
Get phase attribute "Manual Screening":
    String manualScreening = phase.getAttribute("Manual Screening")
    If the return value is "Yes" screening type is manual, otherwise it is automatic.
```

1.3.14 Get the scorecard minimum score using a review [Routine]

```
Get scorecardId from one the review instance: scorecardId = review.getScorecard()
Use Scorecard Management to get the Scorecard instance:
scorecard = ScorecardManager.getScorecard(scorecardId)
    minScore = scorecard.getMinScore()
```

1.3.15 Search all ScreeningTasks for the project [Routine]

```
A screening task is result of automatic screening a submission
    Search all submissions for current project:
        - Use Deliverable Management to search for submission with: ProjectId =
phase.getProject().getId()
    Get the upload ids from the search result Submission[] array:
submission.getUpload().getId()
    Use Auto Screening Management component, call
ScreeningManager.getScreeningTask(uploadIds) to get
    an array of ScreeningTask
```

1.3.16 Registration Phase Handler

```
canPerform() - Can start
- The dependencies are met
- Check phase start date time if exist

Registration phase can start as soon as the dependencies are met
    Use the routine "Check if all dependencies of a phase have stopped".

If phase.calcStartDate() is not null
    Use the routine "Check if a phase start time is reached".
```

canPerform() - Can stop

Registration phase can stop when both of these conditions met:
- The dependencies are met
- The period has passed
- The number of registrations meets the required number.

The dependencies are met
Use the routine "**Check if all dependencies of a phase have stopped**".

The period has passed
Use the "Check if a phase end time is reached" routine.
For "The number of registrations meets the required number":

Get the number of required registrations using the "Registration Number" phase attribute:
String regNumber = phase.getAttribute("Registration Number")
Use Resource Management to search for resources with
Role equals "Submitter"
Project id equals phase.getProject().getId()
Compare the number of resource returned from the search with "regNumber" to see if the condition met.

1.3.17 Submission Phase Handler

canPerform() - Can start
- The dependencies are met
- Check phase start date time if exist

Check the dependencies are met
Use the routine "Check if a phase start time is reached".

If phase.calcStartDate() is not null
Submission phase can start as soon as the start time is reached

canPerform() - Can stop
Submission phase can stop when all of these conditions met:
- The dependencies are met
- The period has passed
- If manual screening is absent, the number of submissions that have passed auto-screening meets the required number;
- If manual screening is required, the number of submissions that have password manual screening meets the required number.

The dependencies are met
Use the routine "**Check if all dependencies of a phase have stopped**".

The period has passed
Use the "Check if a phase end time is reached" routine.

Check screening type:
Use the "**Check screening type**" routine.

Get the required number of submission that pass screening:
Get the "**Submission Number**" phase attribute
String subNumber = phase.getAttribute("Submission Number")

The number of submissions that have passed auto-screening meets the required number:
Get all screening tasks using "**Search all ScreeningTasks for the project**" routine (ScreeningTasks[])
passedNum = 0
For each ScreeningTask instance

```
        If screening.getScreeningStatus().getName()="Passed" Or "Passed with Warning"
            passedNum++
        End If
    End For
    Check if passedNum >= subNumber

The number of submissions that have pass manual screening meets the required number:
Search all screening scorecard for the current phase:
    Locate the next screening phase using "Locating phases" routine. Get the
screeningPhaseId.
    Use the "Search all reviews for a phase base on resource roles" routine.
        Input:
            - phaseId=screeningPhaseId
            - resourceRoleNames="Primary Screener", "Screener"
        Output:
            - screenReviews[]

    Get the screening minimum score:
        Use the "Get the scorecard minimum score using a review" routine:
        (With the first instance in screenReviews[])

    Count the number submissions that pass screening
        passedNum = 0
        For each screeningReview
            If screeningReview.getScore() >= minScore
                passedNum++
            End If
        End For
    Check if passedNum >= subNumber
```

1.3.18 Screening Phase Handler

```
canPerform() - Can start
Screening can start as soon as the dependencies are met.
    Use the routine "Check if all dependencies of a phase have stopped".
If phase.calcStartDate() is not null
    Use the routine "Check if a phase start time is reached".

canPerform() - Can stop
Screening can stop when:
      - The dependencies are met
      - If it's primary screening mode, all submissions that passed auto-screening have
      one screening scorecard committed.
      - If it's individual screening mode, the submission that passed auto-screening has
      one screening scorecard committed.

The dependencies are met
    Use the routine "Check if all dependencies of a phase have stopped".

Determine the screening mode (Primary or individual):
    Lookup resource role id for "Primary Screener" and "Screener" using
ResourceRoleLookupUtility (primaryScreenerId and screenerId)
    Use Resource Management to search for resource with the resource role id =
"primaryScreenerId" and "screenerId"
    If search for "Primary Screener" returns at least one row, the screening mode is
primary
    If search for "Screener" returns at least one row, the screening mode is individual.
    If both return at least one row, throw PhaseHandlingException for inconsistent data.

If it's primary screening mode, all submissions that passed auto-screening have one
screening scorecard committed.
    Search all submissions for current project (submissions[]):
```

Use Deliverable Management to search for submission with: ProjectId = phase.getProject().getId()

Search all screening scorecard for the current phase:
Use the "**Search all reviews for a phase base on resource roles**" routine.
Input:
- phaseId=phase.getId() (current phase id)
- resourceRoleName="**Primary Screener**"
Output:
- screenReviews[]

Check if every submission has an associated screening scorecard
For each submission
Look for its matching scorecard using submission.getId()=screenReview.getSubmission()
End For

Check if all of the screening scorecards are committed
Walk the screenReviews[] and check screenReview.isCommitted()

If it's individual screening mode, the submission that passed auto-screening has one screening scorecard committed.
Search the submission for the current individual screening phase
Search the submitter for this phase
Use the "**Search resource base on resource role names and phase id**" routine.
Input:
- resourceRoleNames: "Submitter"
- phaseId: current phase id (phase.getId())
Output:
- resources[]
If resources[].length <> 1, throw PhaseHandlingException for inconsistence data.
submitter = resource[0]

Search the submission base on the submitter (submissions[])
Use Deliverable Management to search for submission with: ProjectId = phase.getProject().getId()
AND resource Id = submitter.getId()
If submissions[].length <> 1, throw PhaseHandlingException for inconsistence data.
submission=submissions[0]

Use the "**Search all reviews for a phase base on resource roles**" routine.
Input:
- phaseId=phase.getId() (current phase id)
- resourceRoleName="Screener"
Output:
- screenReviews[]
For individual screening mode, each phase will be associated with one submission. So if screenReviews[].length <>1,
throw PhaseHandlingException for inconsistence data.
screenReview=screenReviews[0]

Check if the screening review is associated with the submission
Use submission.getId()=screenReview.getSubmission()

Check if the screening scorecards are committed
Check screenReview.isCommitted()

perform() - Stop
When screening is stopping:
- All submissions with failed screening scorecard scores should be set to the status "Failed Screening"

```
- Screening score for the all submissions will be saved to the submitters' resource
properties named "Screening Score".

    Search all screening scorecard for the current phase:
        Use the "Search all reviews for a phase base on resource roles" routine.
        Input:
            - phaseId=phase.getId() (current phase id)
            - resourceRoleNames="PrimaryScreener", "Screener"
        Output:
            - screenReviews[]

    Search all submissions for current project (submissions[]):
        Use Deliverable Management to search for submission with: ProjectId =
phase.getProject().getId()

    Get the screening minimum score:
        Use the "Get the scorecard minimum score using a review" routine.
        (With the first instance in screenReviews[])

    For each submission
        Find the matching screening review using submission.getId() =
screeningReview.getSubmission()
        Get submission's screening score: screeningScore = screeningReview.getScore()

        Store the screening score for the submission
            Get submitterId: submitterId = submission.getUpload().getOwner()
            Use Resource Management to get submitter (submitter is a resource):
                submitter = ResourceManager.getResource(submitterId)
            Set "Screening Score" property: submitter.setProperty("Screening Score",
screeningScore.toString())
            Save the submitter to the persistence using Resource Management:
ResourceManager.updateResource(submitter, operator)

        If screeningScore<screening minimum score
            Set submission status to "Failed Screening"
                Get submission status id for "Failed Screening" status using
SubmissionStatusLookupUtility class (failedScreeningStatusId)
                Create a new SubmissionStatus with id = failedScreeningStatusId and
name="Failed Screening"
                Set the status for the submission using submission.setSubmissionStatus()
                Update submission using Deliverable Management:
UploadManager.updateSubmission()
        End If
    End for
```

1.3.19 Review Phase Handler

```
canPerform() - Can start
Review can start as soon as the dependency phases have ended
    Use the routine "Check if all dependencies of a phase have stopped".
If phase.calcStartDate() is not null
    Use the routine "Check if a phase start time is reached".

canPerform() - Can stop
- All active submissions have one review scorecard from each reviewer for the phase
- All test case reviewers have one test case upload.

All active submissions have one review scorecard from each reviewer for the phase
    Search all "Active" submissions for current project
        Get submission status id for "Active" status using SubmissionStatusLookupUtility
class (activeStatusId)
        Use Deliverable Management to search for submission with: ProjectId =
phase.getProject().getId()
```

```
        AND submission status id= activeStatusId
    Search the reviewIds using Resource Management
        Search for resource with resource_role_name = "Reviewer" AND phase id =
phase.getId()
        Resource[] reviewers = ResourceManager.search(filter)
    Search all review scorecard for the current phase:
        Use the "Search all reviews for a phase base on resource roles" routine
        Input:
            - resourceRoleName: "Reviewer"
            - phaseId: reviewPhase.getId()
        Output:
            - reviews[]
    For each submission
        Match the submission with its reviews by using submission.getId() and
review.getSubmission()
        Each submission should has the number of review scorecard = the number of
reviewers.
        Match the submission's reviews with the reviewer for consistence check by using
            review.getAuthor() = reviewer.getId()
    End For

All test case reviewers have one test case upload.
    Find all test case reviewer id using InFilter search of Resource Management:
        Lookup resource role id using ResourceRoleLookupUtility
            "Accuracy Reviewer" - accuracyReviewerId
            "Failure Reviewer" - failureReviewerId
            "Stress Reviewer" - stressReviewerId
        Search for resource with: resource_role_id IN [accuracyReviewerId,
failureReviewerId, stressReviewerId] AND
        phase id = phase.getId() (reviewerIds)
    Use DeliverableManagement to search for uploads with the resourceId IN [reviewerIds
array]
    For each reviewerId
        To see if each reviewer has one test case upload
            Match reviewer and his upload by reviewerId = upload.getOwner()
    End For

perform() - Start
All submissions failed automated screening must be set to the status "Failed Screening".

Check screening type using "Check screening type" routine.

Find submissions failed automated screening:
    Get all screening tasks using "Search all ScreeningTasks for the project" routine
(ScreeningTasks[])
    For each screeningTask
        If screeningTask.getScreeningStatus.getName() = "Failed"
            uploadId = screeningTask.getUpload()
            Use Deliverable Management to search for upload with id = uploadId: upload =
UploadManager.getUpload(uploadId)
            submitterId = upload.getOwner()
            Store submitterId in a list (failedSubmitterIds)
        End If
    End For
    Use Deliverable Management to search for submission with resourceId IN
[failedSubmitterIds array](failedSubmissions[])

Update failed submission status to "Failed Screening":
    Get submission status id using for "Failed Screening" status
SubmissionStatusLookupUtility class (failedScreeningStatusId)

    For each failedSubmission
```

```
        Create a new SubmissionStatus with id = failedScreeningStatusId and name="Failed
Screening"
        Set the status for the submission using submission.setSubmissionStatus()
        Update submission using Deliverable Management: UploadManager.updateSubmission()
    End For
```

perform() - Stop

Initial score for the all passed screening submissions will be calculated and saved to the submitters' resource properties named "Initial Score".
Submissions that passed screening will have status "Active" (instead of "Failed Screening")

```
    Search all "Active" submissions for current project
        Use Deliverable Management to search for submission with: ProjectId =
phase.getProject().getId()
        AND submission status = "Active"
    Search the reviewIds using Resource Management
        Search for resource with resource_role_name = "Reviewer" AND phase id =
phase.getId()
        Resource[] reviewers = ResourceManager.search(filter)
    Search all review scorecard for the current phase:
        Use the "Search all reviews for a phase base on resource roles" routine
        Input:
            - resourceRoleName: "Reviewer"
            - phaseId: reviewPhase.getId()
        Output:
            - reviews[]
    For each submission
        Match the submission with its reviews by using submission.getId() and
review.getSubmission()
        Each submission should has the number of review scorecard = the number of
reviewers.
        Get the scores from the reviews scorecard using review.getScore()
        Use Review Score Aggregator to calculate aggregation of initial score (aggScore)
        Get submitterId: submitterId = submission.getUpload().getOwner()
        Use Resource Management to get submitter (submitter is a resource):
            submitter = ResourceManager.getResource(submitterId)
        Set "Initial Score" property: submitter.setProperty("Initial Score",
aggScore.toString())
        Save the submitter to the persistence using Resource Management:
ResourceManager.updateResource(submitter, operator)
    End For
```

1.3.20 Appeals Phase Handler

canPerform() - Can start

```
- Appeals can start as soon as the dependency phases have ended
    Use the routine "Check if all dependencies of a phase have stopped".
If phase.calcStartDate() is not null
    Use the routine "Check if a phase start time is reached".
```

canPerform() - Can stop

```
Appeals can start when:
- The period has passed.
    Use the "Check if a phase end time is reached" routine.
```

1.3.21 Appeals Response Phase Handler

canPerform() - Can start

```
Appeals Response can start as soon as the dependency phases have ended
    Use the routine "Check if all dependencies of a phase have stopped".
If phase.calcStartDate() is not null
    Use the routine "Check if a phase start time is reached".
```

canPerform() - Can stop
The dependencies are met
All appeals are resolved.

The dependencies are met
Use the routine "Check if all dependencies of a phase have stopped".

All appeals are resolved
Find appeals:
Go back to the nearest Review phase.
Get all reviews:
Use "Search resource base on resource role names and phase id" with current phase id and role = "Reviews" to get reviewers
For each reviews
Use review.getAllComments() to get its comments
Use comment.getCommentType().getName() to find comments with type "Appeal" and "Appeal Response"
End For
If the number of Appeal and Appeal Response comments are equal, then all appeals are resolved.

perform() –Stop

When Appeals Response is stopping, all submissions with failed review scores should be set to the status Failed Review.
Overall score for the passing submissions should be calculated and saved to the submitters' resource properties together with their placements. The winner and runner-up should be populated in the project properties.
Submissions that do not win should be set to the status Completed Without Winning.

Update failed review submission status to "Failed Review":
Lookup submission status Id for "Failed Review" status using SubmissionStatusLookupUtility class. (failedReviewStatusId)

Search all "Active" submissions for current project
Use Deliverable Management to search for submission with: ProjectId = phase.getProject().getId()
AND submission status = "Active" (submissions[])

Locate the nearest previous Review Phase (reviewPhase)
Using "Locating phases" routine.

Search all review scorecard for the review phase:
Use the "Search all reviews for a phase base on resource roles" routine
Input:
- resourceRoleName: "Reviewer"
- phaseId: reviewPhase.getId()
Output:
- reviews[]

Get minimum review score (minScore)
Use the "Get the scorecard minimum score using a review"
(with the first instance in reviews[] array)

Use Review Score Aggregator to calculate aggregation of final score (aggScore) and placement (placement)
for each submission.

For each submission
Match the submission with its reviews by using submission.getId() and review.getSubmission()

```
        Each submission should has the number of review scorecard = the number of
reviewers.
        Get the scores from the reviews scorecard using review.getScore()


        If aggScore < minScore
            Create a new SubmissionStatus with id = failedReviewStatusId and
name="Failed Review"
            Set the status for the submission using submission.setSubmissionStatus()
            Update submission using Deliverable Management:
UploadManager.updateSubmission()
        End If

        Get submitterId: submitterId = submission.getUpload().getOwner()
        Use Resource Management to get submitter (submitter is a resource):
            submitter = ResourceManager.getResource(submitterId)
        Set "Final Score" property: submitter.setProperty("Final Score",
aggScore.toString())

        If aggScore>= minScore
            Set "Placement" property: submitter.setProperty("Placement",
placement.toString())

            If placement <> 1
                Lookup submission status Id for "Completed Without Win" status
                    Use SubmissionStatusLookupUtility class. (cwwStatusId)
                Create a new SubmissionStatus with id = cwwStatusId and name="Completed
Without Win"
                Set the status for the submission using submission.setSubmissionStatus()
                Update submission using Deliverable Management:
UploadManager.updateSubmission()
            End If
        End If

        Store the winning submitter in winningSubmitter
        Store the runner up submitter in runnerUpSubmitter

        Save the submitter to the persistence using Resource Management:
ResourceManager.updateResource(submitter, operator)
    End For

Set project properties to store the winner and the runner up:
    Get projectId
        projectId = phase.getProject().getId()
    Gets the project instance using Project Management
        project = ProjectManager.getProject(projectId)

    Get the external reference id of the winner and the runner up, store in the
resource's property
    named "External Reference ID"
        winnerExtId = winningSubmitter.getProperty("External Reference ID")
        runnerExtId = runnerUpSubmitter.getProperty("External Reference ID")

    Set the winner for the project in the project property named "Winner External
Reference ID"
        project.setProperty("Winner External Reference ID", winnerExtId)
    Set the runner up for the project in the project property named "Runner-up External
Reference ID"
        project.setProperty("Runner-up External Reference ID", runnerExtId)

    Update the project
        Use ProjectManager.updateProject() method with updateReason="Update the winner
and runner up."
```

1.3.22 Aggregation Phase Handler

canPerform() - Can start

Aggregation can start as soon as soon as the dependencies are met

Use the routine "Check if all dependencies of a phase have stopped".

If phase.calcStartDate() is not null

Use the routine "Check if a phase start time is reached".

canPerform() - Can stop

The dependencies are met;

The winning submission must have one aggregated review scorecard committed.

The dependencies are met

Use the routine "Check if all dependencies of a phase have stopped".

The winning submission must have one aggregated review scorecard committed.

Search the aggregated review scorecard using the "Search all reviews for a phase base on resource roles" routine.

Input:

- phaseId=current phase id (phase.getId())
- resourceRoleName="Aggregator"

Output: reviews[]

If this array contains more than 1 item, throw PhaseHandlingException for inconsistence.

If it contains 1 item, return true

If it contains 0 items, return false

perform() - start

When Aggregation is starting and Aggregation worksheet is not created, it should be created; otherwise it should be marked uncommitted, as well as the aggregation review comments.

Check if the Aggregation worksheet is created

Search the aggregated review scorecard using the "Search all reviews for a phase base on resource roles" routine.

Input:

- phaseId=current phase id (phase.getId())
- resourceRoleName="Aggregator"

Output: reviews[]

If this array contains more than 1 item, throw PhaseHandlingException for inconsistence.

If it contains 1 item - the Aggregation worksheet is created

aggWorksheet = reviews[0]

If it contains 0 items - the Aggregation worksheet is not created

Create the Aggregation worksheet

If the Aggregation worksheet is not created, **create it using Review Manager**

Search for id of the Aggregator using Final Reviewer

"Search resource base on resource role names and phase id"

Input:

- ResourceRoleName: "Aggregator"
- phaseId: Current phase id (phase.getId())

Output:

resource[]

Create a new Review:

aggReview = new Review()

aggReview.setAuthor(resource[0].getId())

Persist the review

Use ReviewManager.updateReview()

```
    If the Aggregation worksheet is created:
        Mark uncommitted for the worksheet: aggWorksheet.setCommit(false)
        Mark uncommitted for comments:
            Get all comments: aggWorksheet.getAllComments()
            For each comment
                Clear comment extra info: comment.setExtraInfo(null);
            End For
```

1.3.23 Aggregation Review Phase Handler

```
canPerform() - Can start
Aggregation can start as soon as soon as the dependencies are met
    Use the routine "Check if all dependencies of a phase have stopped".

If phase.calcStartDate() is not null
    Use the routine "Check if a phase start time is reached".

canPerform() - Can stop
       The dependencies are met;
       The aggregation review is performed by two reviewers other than the aggregator,
and the winning submitter

    Locate the nearest backward Aggregation phase (aggPhase)
        Use the "Locating phases" routine.

    Search the aggregated review scorecard using the "Search all reviews for a phase
base on resource roles" routine.
    Input:
        - phaseId=Aggregation phase id (aggPhase.getId())
        - resourceRoleName="Aggregator"
    Output: reviews[]
        If it contains 1 item
            aggWorksheet = reviews[0]
        Else
            Throw PhaseHandlingException for inconsistence.
        End If

    Get all comments: aggWorksheet.getAllComments() (comments[])

    Locate the nearest Review phase (reviewPhaseId)
        Use the "Locating phases" routine.
    Use "Search resource base on resource role names and phase id"
    Input:
        - resourceRoleNames: "Reviewer"
        - phaseId: reviewPhaseId
    Output: resources[]

    Locate the winning submitter (winningSubmitter)
        Use Resource Manager, search all submitter for the current project
[submitters[]]
            Lookup resource role Id for "Submitter" using ResourceRoleLookupUtility
(submitterRoleId)
            Search for resource using submitterRoleId AND phase.getProject.getId()
        Find the winning submitter in the submitters[] array:
            The winning submitter has submitter.getProperty("Placement")="1"

    Look in comments[] for comment.getCommentType().getName()="Aggregation Review
Comment"
    and compare comment.getAuthor() with
        In resources[]: resource.getId()
        winningSubmitter.getId()
```

```
    If at least one comment exist for two instance in resources[] array
    AND for the winningSubmitter
        return true (indicates that two reviewers and the winner has reviewed the
aggregation)
    Else
        return false
    End if
```

perform() – stop

When Aggregation Review phase is stopping, if the aggregation is rejected by anyone, another aggregation/aggregation review cycle is inserted.

```
Use similar logic in "canPerform() - Can stop" to get resources[] and comments[]
    Check comment.getExtraInfo() for each comment. Value must be "Approved" or
"Rejected". Otherwise, throws PhaseHandlingException
    If all comments are "Approved"
        return
    Else (there is at least one reject)
        Insert new agg/agg review cycle:
        Get current project by phase.getProject()
            (currentPrj)
        Get current project phases
            Phase[] phases = currentPrj.getAllPhases(). The phases is already sorted in
the right order
            Find the current phase in the array  use phase.getId() (currentPhase)
        Lookup phase type Ids and create phase types use PhaseTypeLookupUtility:
            Create two phase types "Aggregation" and "Aggregation Review"
        Lookup phase status Id and create phase status use PhaseStatusLookupUtility
            Create the "Scheduled" phase status
        Create two Phase instances of
            Current project, "Scheduled" status, type "Aggregation" and "Aggregation
Review" respectively
        Insert the two new phase into "phases" array, after the currentPhase.
        Update the phases in currentPrj instance
            Using currentPrj.addPhase() and removePhase() methods
        Update the phases into persistence using Phase Management component
            Using PhaseManager.updatePhases()
    End if
```

1.3.24 Final Fix Phase Handler

canPerform() - Can start

Final Fix can start as soon as soon as the dependencies are met

```
    Use the routine "Check if all dependencies of a phase have stopped".
```

If phase.calcStartDate() is not null

```
    Use the routine "Check if a phase start time is reached".
```

canPerform() - Can stop

The dependencies are met;
The final fix has been uploaded;

The dependencies are met;

```
    Use the routine "Check if all dependencies of a phase have stopped".
```

Check if the final fix has been uploaded

```
    Use Deliverable Management to search for upload with "Final Fix" type
```

```
    The result array must contain zero or one upload, throws PhaseHandlingException
otherwise.
    If result contains one upload, then the final fix is uploaded

perform() - start
When Final Fix is starting and Final Review worksheet is not created, it should be
created;
otherwise it should be marked uncommitted.  Previous final fix upload will be deleted.

Check if the Final Review worksheet is created
    Search the Final Review worksheet using the "Search all reviews for a phase base on
resource roles" routine.
    Input:
        - Get nearest Final Review phase (frPhase)
        - phaseId=Final Review phase id (frPhase.getId())
        - resourceRoleName="Final Reviewer"
    Output: reviews[]
        If this array contains more than 1 item, throw PhaseHandlingException for
inconsistence.
        If it contains 1 item - the Final Review worksheet is created
            finalWorksheet = reviews[0]
        If it contains 0 items - the Final Review worksheet is not created

Create the Final Review worksheet
    Search for id of the Final Reviewer using
        "Search resource base on resource role name and phase id"
        Input:
            - ResourceRoleName: "Final Reviewer"
            - phaseId: Current phase id (phase.getId())
        Output:
        resource[]

    Create a new Final Review worksheet:
        - Author id: resource[0].getId() (the final reviewer)
        - review.setCommitted(false)
    Store the review into the persistence
        Use ReviewManager.createReview() method

Mark Final Review worksheet uncommitted:
    Use similar logic as in "Check if the Final Review worksheet is created" to get the
    final review worksheet.
    Set committed to false
        finalWorksheet.setCommitted(false)
    Update the worksheet
        Use ReviewManager.updateReview() method

Delete the previous final fix
    Use similar logic as in "Check if the final fix has been uploaded" to get the
finalFix
    Set the finalFix upload status to "Deleted" use finalFix.setUploadStatus()
    Update the finalFix using UploadManager.updateUpload()
```

1.3.25 Final Review Phase Handler

```
canPerform() - Can start
Final Review can start as soon as soon as the dependencies are met
    Use the routine "Check if all dependencies of a phase have stopped".

If phase.calcStartDate() is not null
    Use the routine "Check if a phase start time is reached".

canPerform() - Can stop
```

```
       The dependencies are met;
       The final review is committed by the final reviewer.

The dependencies are met;
    Use the routine "Check if all dependencies of a phase have stopped".

Check if final review is committed by the final reviewer
    Use similar logic as in "Check if the Final Review worksheet is created" to get the
    final review worksheet. (finalReview)
    Check committed value
        Using finalReview.isCommitted()

perform() - stop
When Final Review phase is stopping, if the final review is rejected, another final
fix/review cycle is inserted.

Find out if the final review is rejected
    Use similar logic as in "Check if the Final Review worksheet is created" to get the
    final review worksheet. (finalReview)
    Get all comments
        finalReview.getAllComments() (comments[])
    Search the result array and find the comment with
        comment.getCommentType().getName()="Final Review Comment" (frComment)
    Check if the comment is approve or reject
        If frComment.getExtraInfo()="Rejected" then the final review is rejected.

Insert another final fix/review cycle
    Get current project by phase.getProject()
        (currentPrj)
    Get current project phases
        Phase[] phases = currentPrj.getAllPhases(). The phases is already sorted in the
right order
        Find the current Final Review phase in the array  use phase.getId()
(currentPhase)
    Lookup phase type Ids and create phase types use PhaseTypeLookupUtility:
        Create two phase types "Final Fix" and "Final Review"
    Lookup phase status Id and create phase status use PhaseStatusLookupUtility
        Create the "Scheduled" phase status
    Create two Phase instances of
        Current project, "Scheduled" status, type "Final Fix" and "Final Review"
respectively
    Insert the two new phase into "phases" array, after the currentPhase.
    Update the phases in currentPrj instance
        Using currentPrj.addPhase() and removePhase() methods
    Update the phases into persistence using Phase Management component
        Using PhaseManager.updatePhases()
```

1.3.26 Approval Phase Handler

```
canPerform() - Can start
Approval can start as soon as soon as the dependencies are met
    Use the routine "Check if all dependencies of a phase have stopped".

If phase.calcStartDate() is not null
    Use the routine "Check if a phase start time is reached".

canPerform() - Can stop
       The dependencies are met;
       The approval scorecards are committed;
       All approval scorecards must have passing scores.

The dependencies are met;
```

```
    Use the routine "Check if all dependencies of a phase have stopped".

Check approval scorecards are committed
    Get all approval scorecards using "Search all reviews for a phase base on resource
roles" routine
    Input:
        - phaseId=current phase id (phase.getId())
        - resourceRoleName="Approver"
    Output: approveReviews[]

    Check if the approval scorecards are committed
        Walk the approveReviews[] array and check committed using
approveReview.isCommitted()

Check passing score
    Get minimum review score (minScore)
    Use the "Get the scorecard minimum score using a review"
    (with the first instance in approveReviews[] array)

    Walk the approveReviews[] array and compare score to the minScore.
        Use approveReview.getScore()
```

1.4 Component Class Overview

AbstractPhaseHandler
This abstract class is used as a base class for all phase handlers. This class contains logic in the constructor to load configuration settings for a phase handler. Settings include database connection, email template and email related information.

AggregationPhaseHandler
This class implements PhaseHandler interface to provide methods to check if a phase can be executed and to add extra logic to execute a phase. It will be used by Phase Management component. It is configurable using an input namespace. The configurable parameters include database connection and email sending. This class handles the aggregation phase. If the input is of other phase types, PhaseNotSupportedException will be thrown.
The aggregation phase can start as soon as the dependencies are met and can stop when the following conditions met:

- The winning submission has one aggregated review scorecard committed.

The additional logic for executing this phase is:

- When Aggregation is starting and Aggregation worksheet is not created, it should be created; otherwise it should be marked uncommitted, as well as the aggregation review comments.

AggregationReviewPhaseHandler
This class implements PhaseHandler interface to provide methods to check if a phase can be executed and to add extra logic to execute a phase. It will be used by Phase Management component. It is configurable using an input namespace. The configurable parameters include database connection and email sending. This class handles the aggregation review phase. If the input is of other phase types, PhaseNotSupportedException will be thrown.
The aggregation review phase can start as soon as the dependencies are met and can stop when the following conditions met:

- The dependencies are met
- The aggregation review is performed by two reviewers other than the aggregator, and the winning submitter.

The additional logic for executing this phase is:

- When Aggregation Review phase is stopping, if the aggregation is rejected by anyone, another aggregation/aggregation review cycle is inserted.

AppealsPhaseHandler

This class implements PhaseHandler interface to provide methods to check if a phase can be executed and to add extra logic to execute a phase. It will be used by Phase Management component. It is configurable using an input namespace. The configurable parameters include database connection and email sending. This class handles the appeals phase. If the input is of other phase types, PhaseNotSupportedException will be thrown.
The appeals phase can start as soon as the dependencies are met and can stop when the following conditions met:

- The dependencies are met
- The period has passed.

There is no additional logic for executing this phase.

AppealsResponsePhaseHandler
This class implements PhaseHandler interface to provide methods to check if a phase can be executed and to add extra logic to execute a phase. It will be used by Phase Management component. It is configurable using an input namespace. The configurable parameters include database connection and email sending. This class handles the appeals response phase. If the input is of other phase types, PhaseNotSupportedException will be thrown.
The appeals response phase can start as soon as the dependencies are met and can stop when the following conditions met:

- The dependencies are met
- All appeals are resolved.

The additional logic for executing this phase is:
When Appeals Response is stopping:

- All submissions with failed review scores will be set to the status "Failed Review".
- Overall score for the passing submissions will be calculated and saved to the submitters' resource properties together with their placements. The property names are "Final Score" and "Placement". –
- The winner and runner-up will be populated in the project properties. The property names are "Winner External Reference ID" and "Runner-up External Reference ID".
- Submissions that do not win will be set to the status "Completed Without Win".

ApprovalPhaseHandler
This class implements PhaseHandler interface to provide methods to check if a phase can be executed and to add extra logic to execute a phase. It will be used by Phase Management component. It is configurable using an input namespace. The configurable parameters include database connection and email sending. This class handles the approval phase. If the input is of other phase types, PhaseNotSupportedException will be thrown.
The approval phase can start as soon as the dependencies are met and can stop when the following conditions met:

- The dependencies are met
- The approval scorecards are committed;
- All approval scorecards must have passing scores.

There is no additional logic for executing this phase.

FinalFixPhaseHandler
This class implements PhaseHandler interface to provide methods to check if a phase can be executed and to add extra logic to execute a phase. It will be used by Phase Management component. It is configurable using an input namespace. The configurable parameters include database connection and email sending. This class handles the aggregation phase. If the input is of other phase types, PhaseNotSupportedException will be thrown.
The final fix phase can start as soon as the dependencies are met and can stop when the following conditions met:

- The dependencies are met
- The final fix has been uploaded

The additional logic for executing this phase is:

- When Final Fix is starting and Final Review worksheet is not created, it should be created; otherwise it should be marked uncommitted.
- Previous final fix upload will be deleted.

FinalReviewPhaseHandler
This class implements PhaseHandler interface to provide methods to check if a phase can be executed and to add extra logic to execute a phase. It will be used by Phase Management component. It is configurable using an input namespace. The configurable parameters include database connection, email sending. This class handles the final review phase. If the input is of other phase types, PhaseNotSupportedException will be thrown.
The final review phase can start as soon as the dependencies are met and can stop when the following conditions met:
- The dependencies are met
- The final review is committed by the final reviewer.

The additional logic for executing this phase is:
- When Final Review phase is stopping, if the final review is rejected, another final fix/review cycle is inserted.

RegistrationPhaseHandler
This class implements PhaseHandler interface to provide methods to check if a phase can be executed and to add extra logic to execute a phase. It will be used by Phase Management component. It is configurable using an input namespace. The configurable parameters include database connection, email sending and the required number of registration. This class handles the registration phase. If the input is of other phase types, PhaseNotSupportedException will be thrown.
The registration phase can start whenever the dependencies are met and can stop when:
- The dependencies are met
- The period has passed
- The number of registrations meets the required number.

There is no additional logic for executing this phase.

ReviewPhaseHandler
This class implements PhaseHandler interface to provide methods to check if a phase can be executed and to add extra logic to execute a phase. It will be used by Phase Management component. It is configurable using an input namespace. The configurable parameters include database connection and email sending. This class handles the review phase. If the input is of other phase types, PhaseNotSupportedException will be thrown.
The review phase can start as soon as the dependencies are met and can stop when the following conditions met:
- The dependencies are met
- All active submissions have one review scorecard from each reviewer for the phase;
- All test case reviewers have one test case upload.

The additional logic for executing this phase is:
- When Review phase is starting, all submissions failed automated screening must be set to the status "Failed Screening".

ScreeningPhaseHandler
This class implements PhaseHandler interface to provide methods to check if a phase can be executed and to add extra logic to execute a phase. It will be used by Phase Management component. It is configurable using an input namespace. The configurable parameters include database connection, email sending. This class handles the screening phase. If the input is of other phase types, PhaseNotSupportedException will be thrown.
The screening phase can start as the dependencies are met when the following conditions met:
- The dependencies are met
- If manual screening is not required, all submissions have been auto-screened;
- If manual screening is required, all active submissions have one screening scorecard committed.

The additional logic for executing this phase is:

- When screening is stopping, all submissions with failed screening scorecard scores should be set to the status "Failed Screening". - Screening score for the all submissions will be calculated and saved to the submitters resource properties named "Screening Score".

SubmissionPhaseHandler
This class implements PhaseHandler interface to provide methods to check if a phase can be executed and to add extra logic to execute a phase. It will be used by Phase Management component. It is configurable using an input namespace. The configurable parameters include database connection, email sending and the required number of submissions that pass screening. This class handles the submission phase. If the input is of other phase types, PhaseNotSupportedException will be thrown.
The submission phase can start as soon as the dependencies are met and start time is reached, and can stop when the following conditions met:

- The dependencies are met
- The phase's end time is reached.
- If manual screening is absent, the number of submissions that have passed auto-screening meets the required number;
- If manual screening is required, the number of submissions that have password manual screening meets the required number.

Note that screening phase can be started during a submission phase.
There is no additional logic for executing this phase.

PhaseStatusLookupUtility
This class provides the function to get lookup id from a lookup name of "phase_status_lu" table. Since lookup id/value pairs do not change in the database per installation, this class caches the id/value pairs to minimize queries to the database. This class is used in various methods of PhaseHandler implementations.

PhaseTypeLookupUtility
This class provides the function to get lookup id from a lookup name of "phase_type_lu" table. Since lookup id/value pairs do not change in the database per installation, this class caches the id/value pairs to minimize queries to the database. This class is used in various methods of PhaseHandler implementations.

ProjectInfoTypeLookupUtility
This class provides the function to get lookup id from a lookup name of "project_info_type_lu" table. Since lookup id/value pairs do not change in the database per installation, this class caches the id/value pairs to minimize queries to the database. This class is used in various methods of PhaseHandler implementations.

ResourceRoleLookupUtility
This class provides the function to get lookup id from a lookup name of "resource_role_lu" table. Since lookup id/value pairs do not change in the database per installation, this class caches the id/value pairs to minimize queries to the database. This class is used in various methods of PhaseHandler implementations.

SubmissionStatusLookupUtility
This class provides the function to get lookup id from a lookup name of "submission_status_lu" table. Since lookup id/value pairs do not change in the database per installation, this class caches the id/value pairs to minimize queries to the database. This class is used in various methods of PhaseHandler implementations.

1.5 Component Exception Definitions

ConfigurationException [Custom]
Represents an exception related to loading configuration settings. Inner exception should be provided to give more details about the error. It is used in PhaseHandler implementation classes.

PhaseHandlingException [Custom]
This exception wraps any error occurred during checking or executing a phase. Inner exception should be provided to give more details about the error. It is used in phase handler classes.

PhaseNotSupportedException [Custom]
This exception is thrown by a phase handler when the input phase is not the type the handler can handle. It is used in phase handler classes.

IllegalArgumentException
This exception is used in all classes for invalid arguments. Invalid arguments in this design are usually null objects, empty strings (including all spaces strings), and arrays with null elements.

1.6 Thread Safety

This design is thread safe. Methods in lookup classes are marked as synchronized for thread safety access to the underlying map. Phase handler classes in this component are immutable so they are thread-safe by default. The manager components used in this component are not thread-safe. But since they are used in this component in a single thread manner in a helper class and phase handlers, this should not be a problem.

2. Environment Requirements

2.1 Environment

JDK 1.3

2.2 TopCoder Software Components

- Configuration Manager v2.1.4 – used to read the configuration information.
- DB Connection Factory v1.0 – used to create the connection to the database.
- Base Exception 1.0 – used as the base for all custom exception classes.
- Email Engine 3.0 – used to send email
- Document Generator 2.0 - used to generate email template.
- Phase Management 1.0 – used to search for phases
- Review Management 1.0 – used to search for reviews
- Resource Management 1.0 – used to search for resources.
- Project Management 1.0 – used to search for projects.
- Scorecard Management 1.0 – used to search for scorecards.
- Deliverable Management 1.0 – used to search for uploads and submissions
- Auto Screening Management 1.0 – used to search for auto screening result.
- User Project Data Store 1.0 – used to search for external user and project information.

2.3 Third Party Components

- None

NOTE: The default location for 3rd party packages is ../lib relative to this component installation. Setting the ext_libdir property in topcoder_global.properties will overwrite this default location.

3. Installation and Configuration

3.1 Package Name

com.cronos.onlinereview.phases
com.cronos.onlinereview.phases.lookup

3.2 Configuration Parameters

For a phase handler class:

Parameter	Description	More information
ConnectionFactoryNS	The namespace that contains settings	String – Required

	for DB Connection Factory.	
ConnectionName	The name of the connection that will be used by DBConnectionFactory to create connection. If missing, default connection will be created.	String - Optional
ManagerHelperNamespace	The namespace that will be used by ManagerHelper class. If missing, default namespace of this class will be used.	String - Optional
StartPhaseEmail	Contains configuration properties for sending email at start phase. If missing, email will not be sent in start phase.	String - Optional
EndPhaseEmail	Contains configuration properties for sending email at end phase. If missing, email will not be sent in end phase.	String - Optional
xxPhaseEmail / EmailTemplateSource	Contains the template source settings for start or end phase email.	String - Required
xxPhaseEmail / EmailTemplateName	Contains the template name settings for start or end phase email.	String – Required
xxPhaseEmail / EmailSubject	Settings for the email subject for start or end phase email.	String – Required
xxPhaseEmail / EmailFromAddress	Settings for the from email address for start or end phase email.	String – Required

For a ManagerHelper class:

Parameter	**Description**	**More information**
ProjectManager / ClassName	The full class name of the ProjectManager implementation.	String – Required
ProjectManager / Namespace	The configuration namespace to initialize the ProjectManager instance. If missing the default constructor is used to create the instance.	String – Optional
ScorecardManager / ClassName	The full class name of the ScorecardManager implementation.	String – Required
ScorecardManager / Namespace	The configuration namespace to initialize the ScorecardManager instance. If missing the default constructor is used to create the instance.	String – Optional
ReviewManager / ClassName	The full class name of the ReviewManager implementation.	String – Required
ReviewManager / Namespace	The configuration namespace to initialize the ReviewManager instance. If missing the default constructor is used to create the instance.	String – Optional
ScreeningManager / Namespace	The configuration namespace to initialize the ScreeningManager instance. If missing the default	String – Optional

	constructor is used to create the instance.	
UploadManager / ClassName	The full class name of the UploadManager implementation.	String – Required
UploadManager / UploadSearchBundleName	The SearchBundle name used to search for uploads.	String – Required
UploadManager / SubmissionSearchBundleName	The SearchBundle name used to search for submissions.	String – Required
UploadManager / UploadIdGeneratorName	The name to load IDGenerator for uploads.	String – Required
UploadManager / UploadTypeIdGeneratorName	The name to load IDGenerator for upload types.	String – Required
UploadManager / UploadStatusIdGeneratorName	The name to load IDGenerator for upload statuses.	String – Required
UploadManager / SubmissionIdGeneratorName	The name to load IDGenerator for submissions.	String – Required
UploadManager / SubmissionStatusIdGeneratorName	The name to load IDGenerator for submission statuses.	String – Required
ResourceManager / ClassName	The full class name of the ResourceManager implementation.	String – Required
ResourceManager / ResourceSearchBundleName	The SearchBundle name used to search for resources.	String – Required
ResourceManager / ResourceRoleSearchBundleName	The SearchBundle name used to search for resource roles.	String – Required
ResourceManager / NotificationSearchBundleName	The SearchBundle name used to search for notifications.	String – Required
ResourceManager / NotificationTypeSearchBundleName	The SearchBundle name used to search for notification types.	String – Required
ResourceManager / ResourceIdGeneratorName	The name to load IDGenerator for resources.	String – Required
ResourceManager / NotificationIdGeneratorName	The name to load IDGenerator for notifications.	String – Required

Sample configuration file:

```
<?xml version="1.0" ?>
<CMConfig>
    <!-- Namespace for AggregationPhaseHandler class -->
    <Config name="com.cronos.onlinereview.phases.AggregationPhaseHandler">
        <Property name="ConnectionFactoryNS">
            <Value>Dbconnection.factory</Value>
        </Property>

        <Property name="ManagerHelperNamespace">
            <Value>com.cronos.onlinereview.phases.ManagerHelper</Value>
        </Property>

        <Property name="ConnectionName">
            <Value>dbconnection</Value>
        </Property>

        <Property name="StartPhaseEmail">
            <Property name="EmailTemplateSource">
                <Value>file</Value>
```

```
            </Property>

            <Property name="EmailTemplateName">
                <Value>test_files/AggregationPhaseStartTemplate.txt</Value>
            </Property>

            <Property name="EmailSubject">
                <Value>Aggregation Phase Start</Value>
            </Property>

            <Property name="EmailFromAddress">
                <Value>admin@topcoder.com</Value>
            </Property>
        </Property>

        <Property name="EndPhaseEmail">
            <Property name="EmailTemplateSource">
                <Value>file</Value>
            </Property>

            <Property name="EmailTemplateName">
                <Value>test_files/AggregationPhaseEndTemplate.txt</Value>
            </Property>

            <Property name="EmailSubject">
                <Value>Aggregation Phase End</Value>
            </Property>

            <Property name="EmailFromAddress">
                <Value>admin@topcoder.com</Value>
            </Property>
        </Property>
    </Config>

    <Config name="com.cronos.onlinereview.phases.ManagerHelper">
        <Property name="ProjectManager">
            <Property name="ClassName">
                <Value>com.topcoder.management.project.InformixProjectManager</Value>
            </Property>
            <Property name="Namespace">
                <Value>com.topcoder.management.project.InformixProjectManager</Value>
            </Property>
        </Property>

        <Property name="ScorecardManager">
            <Property name="ClassName">

<Value>com.topcoder.management.scorecard.InformixScorecardManager</Value>
            </Property>
            <Property name="Namespace">

<Value>com.topcoder.management.scorecard.InformixScorecardManager</Value>
            </Property>
        </Property>

        <Property name="ReviewManager">
            <Property name="ClassName">
                <Value>com.topcoder.management.review.InformixReviewManager</Value>
            </Property>
            <Property name="Namespace">
                <Value>com.topcoder.management.review.InformixReviewManager</Value>
            </Property>
        </Property>
```

```
        <Property name="ScreeningManager">
            <Property name="Namespace">

<Value>com.cronos.onlinereview.autoscreening.management.DefaultDBScreeningManager</Value
>
            </Property>
        </Property>

        <Property name="UploadManager">
            <Property name="ClassName">
                <Value>com.topcoder.management.review.InformixReviewManager</Value>
            </Property>

            <Property name="UploadSearchBundleName">
                <Value>UploadSearchBundle</Value>
            </Property>

            <Property name="SubmissionSearchBundleName">
                <Value>SubmissionSearchBundle</Value>
            </Property>

            <Property name="UploadIdGeneratorName">
                <Value>upload_id_seq</Value>
            </Property>

            <Property name="UploadTypeIdGeneratorName">
                <Value>upload_type_id_seq</Value>
            </Property>

            <Property name="UploadStatusIdGeneratorName">
                <Value>upload_status_id_seq</Value>
            </Property>

            <Property name="SubmissionIdGeneratorName">
                <Value>submission_id_seq</Value>
            </Property>

            <Property name="SubmissionStatusIdGeneratorName">
                <Value>submission_status_id_seq</Value>
            </Property>
        </Property>

        <Property name="ResourceManager">
            <Property name="ClassName">

<Value>com.topcoder.management.resource.PersistenceResourceManager</Value>
            </Property>

            <Property name="ResourceSearchBundleName">
                <Value>ResourceSearchBundle</Value>
            </Property>

            <Property name="ResourceRoleSearchBundleName">
                <Value>ResourceRoleSearchBundle</Value>
            </Property>

            <Property name="NotificationSearchBundleName">
                <Value>NotificationSearchBundle</Value>
            </Property>

            <Property name="NotificationTypeSearchBundleName">
                <Value>NotificationTypeSearchBundle</Value>
```

```
            </Property>

            <Property name="ResourceIdGeneratorName">
                <Value>resource_id_seq</Value>
            </Property>

            <Property name="NotificationIdGeneratorName">
                <Value>notification_id_seq</Value>
            </Property>
        </Property>
    </Config>

    <!-- Namespace for DBConnectionFactory component -->
    <Config name="Dbconnection.factory">
        <Property name="connections">
            <Property name="default">
                <Value>dbconnection</Value>
            </Property>

            <Property name="dbconnection">
                <Property name="producer">

<Value>com.topcoder.db.connectionfactory.producers.JDBCConnectionProducer</Value>
                </Property>

                <Property name="parameters">
                    <Property name="jdbc_driver">
                        <Value>com.informix.jdbc.IfxDriver</Value>
                    </Property>

                    <Property name="jdbc_url">
                        <Value>jdbc:informix-
sqli://192.168.1.150:1526/project:INFORMIXSERVER=ol_home</Value>
                    </Property>

                    <Property name="SelectMethod">
                        <Value>cursor</Value>
                    </Property>

                    <Property name="user">
                        <Value>informix</Value>
                    </Property>

                    <Property name="password">
                        <Value>1234</Value>
                    </Property>
                </Property>
            </Property>
        </Property>
    </Config>

    <Config name="com.topcoder.message.email.EmailEngine">
        <Property name="smtp_host_addr">
            <Value>172.16.20.41</Value>
        </Property>

        <Property name="smtp_host_port">
            <Value>25</Value>
        </Property>

        <Property name="username">
            <Value>test</Value>
        </Property>
```

```
        <Property name="password">
            <Value>test</Value>
        </Property>
    </Config>
</CMConfig>
```

Sample template file:

```
%PHASE_TIMESTAMP{Phase timestamp}%

Hello %USER_FIRST_NAME{User first name}% %USER_LAST_NAME{User last name}%,
Handle: %USER_HANDLE{User handle}%

This is the notification on project: %PROJECT_NAME{Project name}%
Version: %PROJECT_VERSION{Project version}%

This is the %PHASE_OPERATION{The phase operation - start/end}% of the %PHASE_TYPE{Phase
type}% phase.
```

3.3 Dependencies Configuration

The connection definitions in DB Connection Factory need to be configured. See the spec of the DB Connection Factory component for details.

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

Load the configuration before using this component.

4.3 Demo

This component is used by Phase Management component. The implementation of the interface PhaseManager will call phase handlers' methods.

In PhaseManager#canStart() method:

PhaseHandler#canPerform() method is called to check if a phase can start.
The passed phase instance status should be 'Scheduled'

In PhaseManager#canEnd() method:

PhaseHandler#canPerform() method is called to check if a phase can end.
The passed phase instance status should be 'Open'

In PhaseManager#start() method:

PhaseHandler#perform() method is called to provide additional starting logic.
The passed phase instance status should be 'Scheduled'

In PhaseManager#end() method:

PhaseHandler#perform() method is called to provide additional ending logic.
The passed phase instance status should be 'Open'

PhaseManager implementations also have to provide a mechanism let user to configure each phase type with the corresponding phase hander.

5. Future Enhancements

Additional phase handlers can be added.
Additional lookup classes can be added.