Auto Screening Management 1.0 Component Specification

1. Design

Automated screening is a tool that screens submissions automatically. Submissions are put in a queue and will be picked up by standalone screening tools that could potentially run from multiple servers. Screening rules are configurable per project category. Once the submission is screened, the results will be logged. This component provides the management API to initiate screening task and to query screening results. Another component will be developed as the screening tool.

The design is pretty straightforward. The *ScreeningTask*, *ScreeningStatus*, *ScreeningResult*, *ScreeningResponse* and *ResponseSeverity* classes represent the data objects associated with the screening management. Each has the default constructor accepting no arguments, as well as getter and setter methods for the attributes in order to be used as bean class. The *ScreeningManager* is the main interface of the component, and the *DefaultDbScreeningManager* provides the default implementation for the Informix database persistence. Users will normally acquire an instance of ScreeningManager through the *ScreeningManagerFactory*.

The use of factory design pattern in ScreeningManagerFactory allows the user to acquire ScreeningManager instances in a single place, instead of using the specific constructors. The concrete ScreeningManager to create depends on the configuration using the Object Factory. It takes into advantage of the fact that the ScreeningManager implementation has its parameters specified in the constructor. Implementations become easy by making use of the API provided by the Object Factory, instead of parsing the configuration properties one by one where some of the properties are optional. In this way, the configuration-related stuffs are centralized in one class, while ScreeningManager implementations do not need to care about configuration support.

The setup requirement of the DefaultDbScreeningManager is to have an unchanged pending status stored in the persistence for the initiation of screening tasks. Other things such as screening statuses and response severity are not cached, since the screening tool may update these dynamically in the persistence. All are retrieved on request.

Additional flexibility such as configurable table names or SQL statements is not considered here. After all, this is a custom component with the DDL scripts provided in the requirements. Fixing all these will make both the implementation and usage simpler.

1.1 Design Patterns

Factory pattern is used in the ScreeningManagerFactory to create different implementations of ScreeningManager. It shields the caller from the concrete class of the implementation.

1.2 Industry Standards

SQL, JDBC

1.3 Required Algorithms

1.3.1 *Using the Object Factory*

Here shows how to use the Object Factory to create ScreeningManager instances. It uses ScreeningManager.class as the key to look for. Using such approach, the complexity of instantiating DBConnectionFactory and IDGenerator is hidden.

```
ObjectFactory objectFactory = new ObjectFactory(new
    ConfigManagerSpecificationFactory(namespace));
```

```
return (ScreeningManager)
    objectFactory.createObject(ScreeningManager.class);
```

1.3.2 *Database Operations*

The tables will be created according to the DDL script provided in "docs/auto_screening.sql". Database transaction should be used to ensure atomicity. Database connections must not be cached within the component. Connection, PreparedStatement and ResultSet should be created for each operation and closed afterwards.

Note that in the following SQL statements, the description behind "--" is just comment and used for explanation. It should not appear in the actual SQL statement.

1.3.2.1 Initiating Screening Task

To retrieve the pending status id, use:

```
SELECT screening_status_id
FROM screening_status_lu
WHERE
name = ?      -- set this parameter to ScreeningStatus.PENDING
```

To check whether a screening task was already initiated with an upload, use:

```
SELECT upload_id
FROM screening_task
WHERE
upload_id = ?      -- set this parameter to upload
```

To insert the screening task into the table, use:

```
INSERT INTO screening_task
(screening_task_id,
 upload_id,
 screening_status_id,
 create_user,
 create_date,
 modify_user,
 modify_date)
VALUES
(?,                 -- set this parameter to unique id generated
 ?,                 -- set this parameter to upload
 ?,                 -- set this parameter to pendingStatusId
 ?,                 -- set this parameter to operator
 ?,                 -- set this parameter to current time
 ?,                 -- set this parameter to operator
 ?)                 -- set this parameter to current time
```

1.3.2.2 Querying Screening Task

To retrieve the screening tasks and their associated screening status with uploads, use:

```
SELECT
a.screening_task_id,
a.upload_id,
```

```
a.screening_status_id,
a.screener_id,
a.start_timestamp,
a.create_user,
a.create_date,
a.modify_user,
a.modify_date,
b.name
FROM screening_task AS a
LEFT JOIN screening_status_lu AS b
ON a.screening_status_id = b.screening_status_id
WHERE
a.upload_id IN (?) -- set this string to uploads joined by comma
```

To retrieve the screening results and its associated screening response and response severity with a screening task id, use:

```
SELECT
a.screening_result_id,
a.screening_response_id,
a.dynamic_response_text,
b.response_severity_id,
b.response_code,
b.response_text,
c.name
FROM screening_result AS a
LEFT JOIN screening_response_lu AS b
ON a.screening_response_id = b.screening_response_id
LEFT JOIN response_severity_lu AS c
ON b.response_severity_id = c.response_severity_id
WHERE
a.screening_task_id = ? -- set this parameter to screening task id
```

1.4 Component Class Overview

ScreeningManagerFactory:

This is a factory class that creates instances of ScreeningManager. The use of factory design pattern allows the user to acquire ScreeningManager instances in a single place, instead of using the specific constructors.

The concrete ScreeningManager to create depends on the configuration using the Object Factory. It takes into advantage of the fact that the ScreeningManager implementation has its parameters specified in the constructor. In this way, the configuration-related stuffs are centralized in one class. Implementations become easy by making use of the API provided by the Object Factory (instead of parsing the configuration properties one by one). ScreeningManager implementations do not need to care about configuration support.

ScreeningManager (interface):

This interface defines the contract for managing screening tasks. It provides the methods to initiate a screening task, and query the screening task and its details. Details refer to the screening results here. Normally, one upload is used to initiate one screening task.

In a sense, implementations should rely on some kind of persistence in order to perform the above tasks. Note that the update of screening task in the persistence (e.g.

add/remove screening results, update screening status, etc) is provided by another component. These two should work together on the same persistence.

DbScreeningManager (abstract class):

This class abstracts the screening manager that uses database as the persistence for the screening tasks. The database connection is made configurable by using the DB Connection Factory component. The connection-related parameters should be provided in the constructor in order to be used by the Object Factory.

Under this abstraction, subclasses don't need to care about how the database connection is specified. Rather, they can just focus on the persistence logic for the operations.

DefaultDbScreeningManager:

The default implementation of the screening manager that uses Informix database as the persistence for the screening tasks. The table structures are provided in the ddl script at "docs/auto_screening.sql".

The implementation uses the ID Generator component to generate unique ids for the initiation of screening tasks. As such, the IDGenerator should be provided in the constructor in order to be used by the Object Factory.

The setup requirement of this screening manager is to have an unchanged pending status in the screening_status_lu table (for the initiation of screening tasks too). Other things such as screening statuses and response severity are not cached, since the other component (using the same persistence) may update these dynamically. All are retrieved on request.

ScreeningTask:

This class encapsulates the task of screening an uploaded submission by a screener. Each screening task is identified by a unique id.

When the screening proceeds, the screening task can have its status changed. Screening results can be added or removed depending on the screener. The auditing fields should be updated accordingly.

ScreeningStatus:

This class represents the screening status of the screening task. Each screening status is identified by a unique id. Each screening task is associated with a screening status.

When the screening task is first initiated, its screening status is set to PENDING.

ScreeningResult:

This class represents a piece of screening result of the screening task. Each screening result is identified by a unique id. Each screening task is associated with multiple screening results.

When the screening task is first initiated, it contains no screening results. Screening results are normally entered by the screener, consisting of the screening response and dynamic text.

ScreeningResponse:

This class represents the screening response associated with a screening result. Each screening response is identified by a unique id. Each screening result is associated with a screening response.

When the submission is being screened, screening response is normally entered by the screener, consisting of response severity, response code and fixed response text.

ResponseSeverity:

This class represents the response severity of the screening response. Each response severity is identified by a unique id. Each screening response is associated with a response severity.

Response severity indicates how serious the response is on the submission. Examples of response severity include "fatal", "warning", etc.

1.5 Component Exception Definitions

ScreeningManagementException:

This is the base exception of this component. All other exceptions should extend from it.

ConfigurationException:

This exception indicates any error related to configuration, such as missing properties, invalid property values, etc. It is thrown from the ScreeningManagerFactory methods.

PersistenceException:

This exception indicates any error related to persistence. It is used to wrap exceptions such as SQLException and IOException. It is thrown from the ScreeningManager methods.

ScreeningTaskAlreadyExistsException:

This exception indicates that a screening task was already initiated with the upload id, when trying to initiate a screening task with it. It is thrown from the initiate method of the ScreeningManager.

ScreeningTaskDoesNotExistException:

This exception indicates that the screening task was not initiated with the upload id, when trying to query the screening task with it. It is thrown from the query methods of the ScreeningManager.

IllegalArgumentException:

This exception can be thrown by methods of all classes when invalid inputs are passed to them, such as null or empty string. Normally an empty string is checked with trimmed result.

1.6 Thread Safety

This component is thread-safe in this release. The DefaultDbScreeningManager is made thread safe by using JDBC transactions. That means enclosing all the JDBC operations done within a method in a transaction. It will be then the responsibility of the database to ensure concurrent access. The component will be able to run even in a distributed environment.

The ScreeningManagerFactory is thread-safe by being stateless. It supports concurrent creation of ScreeningManager instances. The mutable data objects including ScreeningTask, ScreeningStatus, ScreeningResult, ScreeningResponse and ResponseSeverity are not thread-safe themselves, but will be used by the component in a thread-safe manner. The DefaultDbScreeningManager just creates the objects and sets the attributes one by one.

2. Environment Requirements

2.1 Environment

Java 1.4

2.2 TopCoder Software Components

- **Object Factory 2.0** is used to create instances of ScreeningManager from configuration namespace. Implementations of ScreeningManager do not need to care about configuration support.

- **DB Connection Factory 1.0** is used to create database connections for persistence operations.

- **ID Generator 3.0** is used to generate unique ids for the initiation of screening tasks.

- **Base Exception 1.0** is used as a base class for all exceptions. It provides a consistent way to handle the cause exception.

- **Configuration Manager 2.1.4** is used as a dependency of the above components.

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

None

3. Installation and Configuration

3.1 Package Name

com.cronos.onlinereview.autoscreening.management

com.cronos.onlinereview.autoscreening.management.db

3.2 Configuration Parameters

The configuration follows the structure defined in the Object Factory component. No custom property is introduced here. A sample configuration file is provided in "docs/config.xml".

3.3 Dependencies Configuration

DB Connection Factory and ID Generator require its own configuration.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.

- Follow Dependencies Configuration.

- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

None

4.3 Demo

Assume the configuration namespaces in "docs/config.xml" are loaded, and these records are present in the tables. Only relevant columns are shown for demonstration purpose.

screening_task:

screening_task_id	upload_id	screening_status_id	screener_id	start_timestamp
71	51	62	81	2006-07-11 13:27:39

create_user	create_date	modify_user	modify_date
screening_task_create_user_1	2006-07-11 13:45:59	screening_task_modify_user_1	2006-07-11 14:04:19

screening_status_lu:

screening_status_id	name
61	Pending
62	Screening

screening_result:

screening_result_id	screening_task_id	screening_response_id	dynamic_response_text
111	71	101	dynamic_response_text1
112	71	102	dynamic_response_text2
113	71	103	dynamic_response_text3
114	71	104	dynamic_response_text4
115	71	105	dynamic_response_text5

screening_response_lu:

screening_response_id	response_severity_id	response_code	response_text
101	91	response_code_1	response_text1

response_severity_lu:

response_severity_id	name
91	response_severity_lu_name_1

```
// create the ScreeningManager instance with concrete namespace
ScreeningManager manager = ScreeningManagerFactory
                    .createScreeningManager("com.cronos.onlinere
                            view.autoscreening.management");

// or you can create the ScreeningManager instance with default
   namespace
manager = ScreeningManagerFactory.createScreeningManager();

// initiate screening task with upload
manager.initiateScreening(100, "Operator");
```

This will insert a row similar to the following in the **screening_task** table:

screening_task_id	upload_id	screening_status_id	screener_id	start_timestamp
A unique id	100	1	null	null

create_user	create_date	modify_user	modify_date
Operator	6/22/2006 09:00 (current date)	Operator	6/22/2006 09:00 (current date)

4.3.2   *Querying Screening Tasks Without Details*

```
// create the ScreeningManager instance with default namespace
        ScreeningManager manager =
ScreeningManagerFactory.createScreeningManager();
// query screening tasks without details
        ScreeningTask[] screeningTasks =
manager.getScreeningTasks(new long[] {51});
// the screening task id should be 71
        long taskid = screeningTasks[0].getId();
// the upload id should be 51
        long upload = screeningTasks[0].getUpload();
// the screener id should be 81
        long screener = screeningTasks[0].getScreener();
// the start timestamp should be "2006-07-11 13:27:39"
        Date startTimestamp =
screeningTasks[0].getStartTimestamp();
// the creation user should be "screening_task_create_user_1"
        String creationUser = screeningTasks[0].getCreationUser();
// the creation timestamp should be "2006-07-11 13:45:59"
        Date creationTimestamp =
screeningTasks[0].getCreationTimestamp();
// the modification user should be "screening_task_modify_user_1"
        String modificationUser =
screeningTasks[0].getModificationUser();
// the modification timestamp should be "2006-07-11 14:04:19"
        Date modificationTimestamp =
screeningTasks[0].getModificationTimestamp();
// get the screening status of the screening task
        ScreeningStatus screeningStatus =
```

```
screeningTasks[0].getScreeningStatus();
// the screening status id should be 62
        long statusid = screeningStatus.getId();
// the screening status name should be "Pending"
        String name = screeningStatus.getName();
// no results should be returned in this mode, should be an empty
array
        ScreeningResult[] screeningResults =
screeningTasks[0].getAllScreeningResults();
// query non-existent screening tasks without details,
// the returned array should contain null element
        screeningTasks = manager.getScreeningTasks(new long[]
            {1000}, true);
```

4.3.3 Querying Screening Task With Details

```
// create the ScreeningManager instance with default namespace
        ScreeningManager manager =
ScreeningManagerFactory.createScreeningManager();
// query screening task with details
        ScreeningTask screeningTask =
manager.getScreeningDetails(51);
// the screening results should be of length 5
        ScreeningResult[] screeningResults =
screeningTask.getAllScreeningResults();
// the screening result id should be 111
        long id = screeningResults[0].getId();
// the dynamic text should be "dynamic_response_text1"
        String text = screeningResults[0].getDynamicText();
// get the screening response of screening result
        ScreeningResponse screeningResponse =
screeningResults[0].getScreeningResponse();
// the screening response id should be 101
        id = screeningResponse.getId();
// the response code should be "response_code_1"
        String code = screeningResponse.getResponseCode();
// the response text should be "response_text1"
        text = screeningResponse.getResponseText();
// get the response severity of screening response
        ResponseSeverity responseSeverity =
screeningResponse.getResponseSeverity();
// the response severity id should be 91
        id = responseSeverity.getId();
// the response severity name should be
"response_severity_lu_name_1"
                String name = responseSeverity.getName();
```

5. Future Enhancements

None anticipated in the future.