Auto Screening Tool v1.0 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.

The component provides a Command-Line interface so that it can be scheduled and run through the command line. This functionality is provided in the ScreeningTool class and its inner class ScreeningJob. These classes work with the Command Line Utility and Job Scheduler component to run the screening at a specified interval.

The main functionality of the component is encompassed within the Screener class. This class is responsible for coordinating the various sub-components such as the ScreeningTaskDAO to retrieve available screening tasks, the ScreeningTaskChooser to determine which task to take up, and the ScreeningResponseLogger to record the results of screening. The Screener also works with the ScreeningTask, ScreeningData and ScreeningStatus classes which act as Data objects to represent the screening entities.

The actual screening process is in the ScreeningLogic and ScreeningRule classes. The design separates the actual screening activity from the severity and response of performing that activity. This makes it a lot easier to customize the component - for example, in some scenarios, the customer may find that a certain rule is required, while in others, that same rule would only result in a warning; this allows the logic to be changed by simply changing the configuration.

Finally, it may be important to note that the screening rules in the previous version that are presented here have been refactored in some areas. For example: The ZipFile and JarFile rules in the previous version were in separate classes, while the functionality only differed in the file extension that they checked; such rules have been refactored into a single class (in this case, the ArchiveFileRule) with the necessary parameters as configurable. The class documentation for each rule notes which previous version rules it fulfills.

1.1 Design Patterns

Strategy is utilized in the **ScreeningTaskChooser**, **ScreeningTaskDAO, ScreeningResponseLogger, ScreeningResponseDAO** and **ScreeningRule** interfaces. They all define pluggable strategies for performing the tasks for which they were allocated.

1.2 Industry Standards

JDBC 2.0

1.3 Required Algorithms

Please note that most of the algorithms for this component are quite simple and the relevant details are outlined in the Implementation Notes of the Method Documentation.

1.3.1 Retrieving the Screening Task

The following SQL Query is used:

```
SELECT
```

```
        project.project_id, project.project_category_id,
        upload.upload_id, upload.parameter,
        resource_info.value,
        screening_status_lu.name,
        screening_task.screening_task_id,
        screening_task.screener_id, screening_task.start_timestamp,
        screening_task.create_user, screening_task.create_date,
        screening_task.modify_user, screening_task.modify_date
FROM screening_task
    INNER JOIN screening_status_lu ON screening_task.screening_status_id
        = screening_status_lu.screening_status_id
    INNER JOIN upload ON screening_task.upload_id = upload.upload_id
    INNER JOIN project ON upload.project_id = project.project_id
    INNER JOIN resource_info ON upload.resource_id = resource_info.resource_id
WHERE resource_info.resource_info_type_id=1 // this is the id for a resource info type of
"External Ref Id"
```

Note that the WHERE clause may vary depending on the type of parameters that were provided.

If screener id is specified, append `" AND screening_task.screener_id=" + screenerId`.

If screening status id is specified, append `" AND screening_task.screening_status_id=" + statusId`.

1.3.2 *Concurrency Concerns When Updating the Screening Task*

One of the concerns in this component is the issue where multiple screeners are running and modifications occur to the Screening Task table at the same time. This may occur if 2 concurrent screeners request a list of available data, and then decide to screen the same task. It would result in a conflict, because they would both set the Screening Task to their own respective id and screen it themselves. It may be solved by adding an additional WHERE clause to the update SQL statement. We only allow the following update cases:

1. Update screening status from ‘pending’ to ‘screening’.
2. Update screening status from ‘screening’ to ‘pending’ for rollback.
3. Update screening status from ‘screening’ to ‘failed’.
4. Update screening status from ‘screening’ to ‘passed’.
5. Update screening status from ‘screening’ to ‘passed with warning’.

The following SQL with additional WHERE clause is used:

```
UPDATE screening_task
SET
        screening_status_id=?, screener_id=?, start_timestamp=?,
        create_user=?, create_date=?, modify_user=?, modify_date=?

WHERE screening_task_id=? AND screening_status_id = 1 or 2 // 1 is the screen status for
pending and 2 is the screening status for screening.
```

If the new screening status is ‘screening’, we use 1 in the above WHERE clause; otherwise we use 2.

By doing this, if 2 near-simultaneous updates occurred, only the first update would get in due to the additional WHERE clause. The second one would not update because the screen_status would be set to SCREENING by the first update and the screenerId would also be set by the first update.

The DAO which was not able to perform an update should notify the screener by throwing the UpdateFailedException.

This is based on the Optimistic Concurrency Lock approach that aims on detecting any concurrent updates that occur and preventing it. Once the screener is informed of the concurrent update via UpdateFailedException, then the screener may choose to select a different task for screening.

1.3.3 Additional Logging

All the rule implementations are expected to perform additional logging. The logs should be acquired from the LogFactory upon the start of the screen method (and cleanup, where a method implementation for cleanup is present) via the following:

```
Log log = LogFactory.getInstance().getLog(this.class.getName());
```

The developer may choose to cache this in a class variable if desired.
The following information should be logged at the start of the method call to screen:

- The id of the screening task that was selected.
- The ScreenerId of the screener that is performing this task.

The following information should be logged at the end of the method call to screen:

- Each ScreeningResult message.
- The status (success/failure) of the ScreeningResultMessage.
- The stack trace of any exceptions that were encountered.

1.4 Component Class Overview

[com.cronos.onlinereview.autoscreening.tool]

ScreeningTool

This class provides the command-line interface for the component. It is responsible for parsing the configurable command-line parameters and initializing the component to perform screening using the switches that were provided. It utilizes the TC Job Scheduler to make sure that the screening process runs at regular intervals.

ScreeningJob

This is an inner class that is used to adapt the Screener class and allow it to be run by the Job scheduler. The run method of this class simply delegates to the screen method of the screener. It will always return "screening" as status.

Screener

This is the main workhorse of the component. It is responsible for retrieving the screening tasks from the data source, and determining the screening task to utilize. It will then perform screening on the chosen task; the rules of the screening are based on the project related to the Screening task and the Project category.

ScreeningLogic

This class is used to encapsulate screening logic. It separates itself from the actual screening rule in order to disassociate the severity of a rule from the procedure that is needed to check it. This allows for a more flexible configuration if the client decides to change the severity of a rule (for example, a rule mandating presence of Sequence Diagrams may cause failure in one situation, but may only cause a warning in a different situation).

ResponseLevel

This is a type-safe enum that is used to represent the severity of a response according to the results of applying a screening rule.

RuleResult

This is a data class that represents the result of screening. It indicates whether the task satisfied the requirements of the rule (successful if this happens), and an optional message providing more details as to how the task satisfied (or failed) the rule requirements. It may also contain a throwable in the case that the screening rule ended in an error.

ScreeningTask

This is a simple data class that encapsulates the different information related to a Screening Task. It contains only getters and setters.

ScreeningData

This is a data class that is used to provide additional screening data regarding a screening task. It contains only getters and setters.

ScreeningStatus

This is a typesafe enum that provides all the possible Screening Status that a Screening task can undergo. It is possible to retrieve the different constants via their identifier (thorugh the static Enum.getEnumByOrdinal) or name (static Eum.getEnumByStringValue)

ScreeningTaskChooser [interface]

This is an interface that describes a strategy for determining the ScreeningTask to screen. Implementations may use the different attributes of the ScreeningTask (such as to prioritize certain Projects or ProjectCategories).

ScreeningTaskDAO [interface]

This is a DAO that performs some data access operations on the backing persistent store. The interface that is provided is minimalistic, and doesn't encapsulate a full management API. The interface is used simply to satisfy all the needs of the screening tool (for actual DAO Screening Task management, the Screening Management component is suggested)

ScreeningResponseLogger [interface]

This is an interface that is used to describe the contract for logging the screening responses. Implementations are expected to record the result of performing the screening rule, as defined by the screening logic rules.

ScreeningRule [interface]

This is the interface that defines the screening logic to be used in a particular situation. The results of screening are encapsulated within a RuleResult.

[com.cronos.onlinereview.autoscreening.tool.chooser]

EarliestTaskChooser

This is the default implementation of the TaskChooser interface. It chooses a task according to the task with the earliest startTimestamp.

[com.cronos.onlinereview.autoscreening.tool.logger]

DAOLogger

This implementation of the ScreeningResponseLogger allows the screening logic to be logged into a datastore. The logic for actually accessing the datastore is delegated to a Data Access Object.

[com.cronos.onlinereview.autoscreening.tool.informix]

InformixTaskDAO

This implementation of the ScreeningTaskDAO provides an Informix database as the backing datastore.

InformixResponseDAO

This implementation of the ResponseTaskDAO provides an Informix database as the backing datastore.

[com.cronos.onlinereview.autoscreening.tool.rule]

ArchiveFileRule

This rule checks to see whether the file is a valid archive, and unzips it for further rules to process.

ComponentSpecificationRule

This rule checks if a component specification exists.

FileExtensionRule

This rule determines whether files with a given extension exist within the submission. It is used to encapsulate JavaSourceCodeRule and CSharpSourceCodeRule from the previous version.

DirectoryStructureRule

This rule checks if the submission conforms to proper directory structure. This class encapsulates the CSharpDevDirRule, JavaDevDirRule, DesignDirRule, TestLogRule from the initial version. With the correct configuration, the scenarios that needed validation could be validated using the Directory Validator component.

CheckStyleRule

This rule runs checkstyle on the Java source code.

ValidationLoggerImpl [package-private]

This is a ValidationLogger that is used as an inner class of the DirectoryStructureRule. It will generate a RuleResult for each rule validation result that it encounters. It will create an invalid RuleResult for each call to the validateFailed method.

PersonalInfoRule

This rule ensures that the submission has no Personal Information. The submitter information is retrieved from the USer Data Store, and the submission is checked for any references to the submitter info.

PoseidonExtractRule

This rule checks if a Poseidon file exists. If it does, it will extract the proj3 file and make it available to other rules for additional checking.

PoseidonDiagramRule

This rule checks an extracted PROJ3 file for the presence of any necessary diagrams. The Topcoder XMI Parser is used to detect the presence of these diagrams.

1.5 Component Exception Definitions

Standard Exceptions

IllegalArgumentException

This exception is thrown when null, empty String or otherwise invalid parameters are passed on to any of the methods in this component. **Note that empty Strings also include a String full of whitespace.**

Custom Exceptions

DAOException

This exception is thrown by the DAOs to signify a problem with accessing the data store.

ConfigurationException

This exception is thrown when a problem occurs while configuring this component.

ScreeningException

This exception is thrown when a problem occurs while performing the screening.

UpdateFailedException

This exception is thrown if the DAO has failed to successfully update the Screening Task because it has already been modified by another concurrently running Screening Task.

1.6 Thread Safety

This component is meant to be run as a command-line application. There will be only a single thread that will be running for the lifetime of this component, so thread safety is not required for any of the classes.

The developer should, however, note that concurrent instances of this component may be run from different machines at the same time and access the same data store. The developer and stress tester should ensure that no issues arise from concurrent access to the data store.

The issue of concurrent access is when two screening tasks will pick the same task and attempt to register this with the database at the same time. This scenario is avoided as outlined in section 1.3.2 of the algorithm section. The Stress tester should write tests that duplicate this scenario and ensure that no 2 tasks are picked by different screeners at the same time.

The Thread-Safety notes of each class can be found in the Class Documentation Overview.

2. Environment Requirements

2.1 Environment

- Java 1.4

2.2 TopCoder Software Components

- **Base Exception 1.0** is used as the base exceptions for the custom exceptions in this component.
- **DB Connection Factory 1.0** is used to retrieve connections from the database where necessary.
- **Configuration Manager 2.1.4** is used to provide configuration support for some of the more complex configurations specified.
- **Object Factory 2.0** is used in conjunction with Configuration Manager to ease development when configuration custom items.
- **Directory Validator 1.0** is used to provide directory validation for the Screening Rules.
- **Compression Utility 1.0** is used to uncompress the ZUML, ZIP and JAR submissions that are provided for screening.
- **Magic Numbers 1.0** is used to determine that the contents of some files are of the appropriate type.
- **Executable Wrapper 1.0** is used to run checkstyle to check the submission coding style.
- **User Data Store 1.0** is used to retrieve the submissions Personal Information to check whether any references exist within the submission.
- **XMI Parser 1.0** is used to analyze a design submission and determine whether it contains the necessary diagrams.
- **Logging Wrapper 1.2** is used to provide additional logging at the Rule level.
- **User Project Data Store 1.0** is used to retrieve the user personal information in order to implement the Personal Information rule.
- **Job Scheduler 1.0** is used to schedule the execution of the Screening task.
- **ID Generator 3.0** is used to generate unique ids for the Screening Response entries.

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.

2.2.1 Unused Components

The Following components were not used because they are not needed to fulfill the requirements:

- Data Validation
- Generic Event Manager
- File Class

2.3 Third Party Components

- There are no direct 3rd party dependencies. See the 3rd party component dependencies of the components that this component is based on for indirect dependencies.

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.autoscreening.tool
com.cronos.onlinereview.autoscreening.tool.rules
com.cronos.onlinereview.autoscreening.tool.informix
com.cronos.onlinereview.autoscreening.tool.chooser
com.cronos.onlinereview.autoscreening.tool.logger

3.2 Configuration Parameters

Configuration Manager Parameters:

Configuration Parameters for Screener:

Parameter	Description
objectFactoryNamespace	The String value that will be provided to the ConfigManagerSpecificationFactory in order to initialize the ObjectFactory to produce the DAO, Logger, and ScreeningLogic instances. **Required.**
screeningTaskDAO	This property will contain subproperties indicating how to instantiate the ScreeningTaskDAO. **Required.**
screeningTaskDAO.class	The value of this property will contain the name of the class to provide to the ObjectFactory to instantiate the screening task DAO. **Required.**
screeningTaskDAO.identifier	The value of this property will contain the identifier to provide to the ObjectFactory to

	instantiate the screening task DAO. **Optional.**
screeningResponseLogger	This property will contain subproperties indicating how to instantiate the ScreeningTaskLogger. **Required.**
screeningResponseLogger.class	The value of this property will contain the name of the class to provide to the ObjectFactory to instantiate the screening task logger. **Required.**
screeningResponseLogger.identifier	The value of this property will contain the identifier to provide to the ObjectFactory to instantiate the screening task logger. **Optional.**
taskChooser	This property will contain subproperties indicating how to instantiate the ScreeningTaskChooser. **Required.**
taskChooser.class	The value of this property will contain the name of the class to provide to the ObjectFactory to instantiate the screening task chooser. **Required.**
taskChooser.identifier	The value of this property will contain the identifier to provide to the ObjectFactory to instantiate the screening task chooser. **Optional.**
remoteFileUpload	This property will contain subproperties indicating how to instantiate the RemoteFileUpload. **Required.**
fileUpload.class	The value of this property will contain the name of the class to provide to the ObjectFactory to instantiate the RemoteFileUpload. **Required.**
fileUpload.identifier	The value of this property will contain the identifier to provide to the ObjectFactory to instantiate the RemoteFileUpload. **Optional.**
projectRules	This property will contain subproperties indicating how to create and map the screening rules for a particular project id. **Required.**
projectRules.<<arbitraryName>>	This property will contain the information for a single project id. **Optional.**
projectRules.<<arbitraryName>>.id	This property will contain the project id for which the screening rules correspond to. **Required.**
projectRules.<<arbitraryName>>.screeningLogic List	The values of this property will indicate the screening logic to use, and the order in which they should be performed. The configuration will look for a subproperty under screeningLogic corresponding to each value in the provided list. **Required.**
projectRules.<<arbitraryName>>.screeningLogic .<<logicName>>	The name of this property should correspond to a value within the screeningLogicList. It contains subproperties indicating how to instantiate and categorize the screening logic. **Required for each value in the**

	screeningLogicList
projectRules.<<arbitraryName>>.screeningLogic.<<logicName>>.ruleClass	The value of this property will contain the name of the class to provide to the ObjectFactory to instantiate the ScreeningRule. **Required.**
projectRules.<<arbitraryName>>.screeningLogic.<<logicName>>.ruleIdentifier	The value of this property will contain the identifier to provide to the ObjectFactory to instantiate the ScreeningRule. **Optional.**
projectRules.<<arbitraryName>>.screeningLogic.<<logicName>>.succeedSeverity	The value of this property will indicate the severity of the rule in the case of success. **Required.**
projectRules.<<arbitraryName>>.screeningLogic.<<logicName>>.failSeverity	The value of this property will indicate the severity of the rule in the case of failure. **Required.**
projectRules.<<arbitraryName>>.screeningLogic.<<logicName>>.errorSeverity	The value of this property will indicate the severity of the rule in the case of an error. **Required.**
projectRules.<<arbitraryName>>.screeningLogic.<<logicName>>.succeedCode	The value of this property will indicate the response code of the rule in the case of success. **Optional.**
projectRules.<<arbitraryName>>.screeningLogic.<<logicName>>.errorCode	The value of this property will indicate the response code of the rule in the case of error. **Optional.**
projectRules.<<arbitraryName>>.screeningLogic.<<logicName>>.failCode	The value of this property will indicate the response of the rule in the case of failure. **Optional.**
categoryRules	This property will contain subproperties indicating how to create and map the screening rules for a particular project category id. **Required.**
categoryRules.<<arbitraryName>>	This property will contain the information for a single project category id. **Optional.**
categoryRules.<<arbitraryName>>.id	This property will contain the project id for which the screening rules correspond to. **Required.**
categoryRules.<<arbitraryName>>.screeningLogicList	The values of this property will indicate the screening logic to use, and the order in which they should be performed. The configuration will look for a subproperty under screeningLogic corresponding to each value in the provided list. **Required.**
categoryRules.<<arbitraryName>>.screeningLogic.<<logicName>>	The name of this property should correspond to a value within the screeningLogicList. It contains subproperties indicating how to instantiate and categorize the screening logic. **Required for each value in the screeningLogicList**
categoryRules.<<arbitraryName>>.screeningLogic.<<logicName>>.ruleClass	The value of this property will contain the name of the class to provide to the ObjectFactory to instantiate the ScreeningRule. **Required.**
categoryRules.<<arbitraryName>>.screeningLogic.<<logicName>>.ruleIdentifier	The value of this property will contain the identifier to provide to the ObjectFactory to instantiate the ScreeningRule. **Optional.**

categoryRules.<<arbitraryName>>.screeningLogic.<<logicName>>.succeedSeverity	The value of this property will indicate the severity of the rule in the case of success. **Required.**
categoryRules.<<arbitraryName>>.screeningLogic.<<logicName>>.failSeverity	The value of this property will indicate the severity of the rule in the case of failure. **Required.**
categoryRules.<<arbitraryName>>.screeningLogic.<<logicName>>.errorSeverity	The value of this property will indicate the severity of the rule in the case of an error. **Required.**
categoryRules.<<arbitraryName>>.screeningLogic.<<logicName>>.succeedCode	The value of this property will indicate the response code of the rule in the case of success. **Optional.**
categoryRules.<<arbitraryName>>.screeningLogic.<<logicName>>.errorCode	The value of this property will indicate the response code of the rule in the case of error. **Optional.**
categoryRules.<<arbitraryName>>.screeningLogic.<<logicName>>.failCode	The value of this property will indicate the response of the rule in the case of failure. **Optional.**

3.3 Dependencies Configuration

The dependency components should be configured according to their documentation.

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

Follow the “Demo”.

4.3 Demo

4.3.1 Command-Line Demo

The demo is meant to be run from the command line, and a command line similar to this one may be used:

java ScreeningTool -screenerId 23 -interval 30000 -configNamespace com.cronos.autoscreening

This would run the ScreeningTool with an id of ‘23’, it would run every 30000 milliseconds (30 seconds), and use the configuration specified in “com.cronos.autoscreening”.

4.3.2 Persistence Demo

```
// Create the ScreeningTaskDAO
ScreeningTaskDAO dao = new InformixTaskDAO(new DBConnectionFactoryImpl(DB_NAMESPACE),
    new DBUserRetrieval(DB_NAMESPACE));
// connection name can be specified explictly
```

```java
dao = new InformixTaskDAO(new DBConnectionFactoryImpl(DB_NAMESPACE),
    "informix_connection", new DBUserRetrieval(DB_NAMESPACE));

// Load all pending tasks.
ScreeningTask[] result = dao.loadScreeningTasks(null, ScreeningStatus.PENDING);

// Retrieve the first task and update it.
ScreeningTask chosenTask = result[0];

// set the screener id and screening status
chosenTask.setScreenerId(12);
chosenTask.setScreeningStatus(ScreeningStatus.SCREENING);

// Update it in the DAO.
try {
    dao.updateScreeningTask(chosenTask);
} catch (UpdateFailedException e) {
    // Find another task to update. Because it probably has already been
    // modified by another concurrently running Screening Task.
    // ...
}

// Create screening response DAO.
ScreeningResponseDAO resDao = new InformixResponseDAO(new DBConnectionFactoryImpl(
    DB_NAMESPACE), IDGeneratorFactory.getIDGenerator("screening_result_id_seq"));
// connection name can be specified explicitly
resDao = new InformixResponseDAO(new DBConnectionFactoryImpl(DB_NAMESPACE),
    "informix_connection", IDGeneratorFactory.getIDGenerator("screening_result_id_seq"));

// Create a new Screening Response.
ScreeningResponse response = new ScreeningResponse();
response.setScreeningTaskId(chosenTask.getId());
response.setResponseCodeId(2);
response.setDetailMessage("detail message");
response.setCreateDate(new Date());
response.setCreateUser("operator");
response.setModificationDate(new Date());
response.setModificationUser("operator");

// Store it in the database.
resDao.createScreeningResponse(response);
```

5. Future Enhancements

Additional ScreeningRule and ScreeningChooser implementations.