Scorecard Management 1.0 Component Specification

1. Design

A scorecard is a template to produce reviews. The scorecard hierarchy consists of groups, sections and questions. Weight can be assigned on each level to control the contribution to the overall score of the scorecard. Different question types are supported and each question can be attached with a document, for example, a test log. A scorecard has minimum and maximum passing score, and has type, status and category attributes. A simple versioning mechanism will be built, where only scorecards not currently in use can be modified.

The component provides the management functionalities such as create, update or search scorecards. The scorecard persistence logic is pluggable.

The main class of this component is ScorecardManagerImpl. This class provides methods to create, update, retrieve and search for scorecards. These operations process the scorecard as a whole, which mean they also process groups, sections and questions belong to a scorecard. For example, the creation of a scorecard may involve many database tables and many rows may be added.

The ScorecardPersistence interface defines the contract for persistence implementations. This allows future implementations to plug into this component. For this version, Informix database is supported. The manager loads the persistence implementation using settings from a configuration namespace.

Scorecard searching function uses SearchBuilder component. By using this component, various types of search condition can be created. The supported search criteria include: scorecard type, scorecard status, project, project category, scorecard name and scorecard version and any combination of them.

The ScorecardValidator interface defines the contract for scorecard validator implementation. This allows future validator to plug-into this component. A default validator is provided with this component with some basic rules to validate the scorecard tree to make sure it conforms to the database schema. Validator settings are also configurable in the ScorecardManagerImpl class.

This design provides some enhancement features such as:

- Pluggable validator: Future implementations can add have more complex rules to validate scorecards.

Note: This design will be separated into two development projects. The first project implements main and validation packages. The second project implements the persistence package.

1.1 Design Patterns

Strategy pattern is used in ScorecardValidator and ScorecardPersistence interfaces. This allows plugging in different implementations of scorecard validator and scorecard persistence.

Facade pattern is used in ScorecardManagerImpl. This class utilizes validator and persistence implementations.

1.2 Industry Standards

JDBC, XML, SQL

1.3 Required Algorithms

1.3.1 Common logic for creating/updating/retrieving scorecards:

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 create/update method, 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 items.
If error occurred, call rollback, and then throw PersistenceException
that wrap the error.
Call commit if everything is fine.
Close the connection in finally block.
```

1.3.2 Check scorecard version format

This logic belong to DefaultScorecardValidator# validateScorecard(Scorecard scorecard) method:
The check logic is simple, so I just provide the check conditions. Throw ValidationException if any condition fails.

```
Length is smaller than or equal to 16.
Contain all number (the dot character is also allowed)
Contains at least one numeric character
The dot character cannot be the first or last character in the version
string.
```

1.3.3 Determine if a scorecard is in use

A scorecard is considered in use when it is referenced in the phase_criteria table. The following SQL command check for this:

```
SELECT COUNT(*) FROM phase_criteria
JOIN phase_criteria_type_lu ON
phase_criteria.phase_criteria_type_id =
phase_criteria_type_lu.phase_criteria_type_id
WHERE phase_criteria_type_lu.name="Scorecard ID"
AND phase_criteria.parameter =?
```

The "?" parameter will be the id of the scorecard. If the query returns one or more rows, then the scorecard is in use.

When retrieving scorecard based on a list of ids, the command should be:

```
SELECT parameter FROM phase_criteria
JOIN phase_criteria_type_lu ON
phase_criteria.phase_criteria_type_id =
phase_criteria_type_lu.phase_criteria_type_id
WHERE phase_criteria_type_lu.name="Scorecard ID"
AND phase_criteria.parameter IN (id1, id2,.., idN);
```

The returned rows contains the id of the in use scorecards, store these values to an array and use them to set scorecard "in use" value using scorecard.setInUse() method. See "*Retrieve scorecard/group/section*" for more information.

1.3.4 Create scorecard/group/section

The creation of these instances is similar and related to each other. They together help persist the scorecard tree in the database. The creation of questions is simple and described in the class documentation.

InformixScorecardPersistence#createScorecard() method:

```
Generate a new id for the scorecard using IdGenerator
Create a new row for the scorecard in its corresponding table (scorecard)
Create an InformixGroupPersistence instance
For each group in the scorecard
        Generate a new id for the group
        Use the created InformixGroupPersistence instance to create the group
End For
Assign the generated id back to the scorecard instance for consistency.
```

InformixGroupPersistence#createGroup() method:

```
Generate a new id for the group using IdGenerator
Create a new row for the group in its corresponding table (scorecard_group)
Create an InformixSectionPersistence instance
For each section in the group
        Generate a new id for the section
        Use the created InformixSectionPersistence instance to create the
section
End For
Assign the generated id back to the group instance for consistency.
```

InformixSectionPersistence#createSection() method:

```
Generate a new id for the section using IdGenerator
Create a new row for the section in its corresponding table (scorecard_section)
Create an InformixQuestionPersistence instance
For each question in the section
        Generate a new id for the question
        Use the created InformixQuestionPersistence instance to create the
question
End For
Assign the generated id back to the section instance for consistency.
```

1.3.5 Update scorecard/group/section

The modification of these instances is similar and related to each other. They together help persist the scorecard tree in the database. The update of question is simple and described in the class documentation.
Note that when updating group and section, the removed section ids and question ids are collected to two lists and will be deleted in the end of update scorecard method.

InformixScorecardPersistence#updateScorecard() method
:

```
Get the scorecard from the persistence using getScorecard(id) method
(oldScorecard)
       It will be used to find out removed items from the updated instance.

Check if the scorecard id exists in the database, if not, throw exception
Determine if the scorecard is in use
       Use the “Determine if a scorecard is in use” routine to determine if the
current scorecard is in use or not.
       If it is in use throws exception indicates that the scorecard is in use
and cannot be modified.
```

Query ‘scorecard’ table to get the current version of the scorecard
Increase the minor version by 1 (the number behind the last dot character – if there is no dot, append a dot first)
(The version format is already validated using the logic in DefaultScorecardValidator)

Find the added group instance:
        Added group instances are instance that id equals 0
Find the removed group instance:
        Get a list of group ids previously belong to the scorecard using “oldScorecard” instance
        Compare to the current group ids to see what was removed
Find the updated group instance:
        The instances that do not belong to the two above
Create new group instances in the database using InformixGroupPersistence#createGroup()
Update the rest by using InformixGroupPersistence#udpateGroup(). Pass two empty lists named deletedSectionIds and deletedQuestionIds to this method.
Collect deletedSectionIds and deletedQuestionIds from updateGroup() calls.

Delete removed group instances in the database using InformixGroupPersistence#deleteGroups()
Delete removed section instances in the database using InformixSectionPersistence#deleteSections(). The deletedSectionIds is collected from updateGroup() calls.
Delete removed question instances in the database using InformixQuestionPersistence#deleteQuestions().The deletedQuestionIds is collected from updateGroup() calls.

InformixGroupPersistence#updateGroup() method:

Check if the group id exists in the database, if not, throw exception
Update group data in the corresponding table (scorecard_group)

Find the added section instance:
        Added section instances are instance that id equals 0
Find the removed section instance:
        Get a list of section ids previously belong to the group using “oldScorecard” instance
       Compare to the current section ids to see what was removed
Find the updated section instance:
        The instances that do not belong to the two above
Create new section instances in the database using InformixSectionPersistence#createSection()
Save the deleted section Ids to the “deletedSectionIds” output parameter.

Update the rest using InformixSectionPersistence#udpateSection().Pass an empty list named deletedQuestionIds to this method.
Collect deleted question Ids from updateSection() calls.

Save the deleted question Ids to the “deletedQuestionIds” output parameter.

InformixSectionPersistence#updateSection() method:

Check if the section id exists in the database, if not, throw exception
Update section data in the corresponding table (scorecard_section)

Find the added question instance:
        Added question instances are instance that id equals 0
Find the removed question instance:
        Get a list of question ids previously belong to the section using

```
“oldScorecard” instance
        Compare to the current question ids to see what was removed
Find the updated question instance:
        The instances that do not belong to the two above

Create new question instances in the database using
InformixQuestionPersistence#createQuestion()

Update the rest using InformixQuestionPersistence#udpateQuestion()

Save the deleted question Ids to the “deletedQuestionIds” output parameter.
```

1.3.6 Retrieve scorecard/group/section

Because of performance reason when retrieving a scorecard, we need to reduce the queries to the database as much as possible. The persistence layer has two getScorecard() method. One method receive the scorecard id as parameter, the other method receive a list of scorecard ids to retrieve all of them at the same time.

The following is the algorithm to retrieve scorecards using a list of scorecard ids. The method that receives a single scorecard id can pass a list contains a single id to this method.

```
Get all scorecard instances for the given scorecard ids.

Get the ids of in use scorecard using “Determine if a scorecard is in
use”.(inUseIds)

SELECT scorecard_id, status.scorecard_status_id, type.scorecard_type_id,
project_category_id, scorecard.name, version, min_score, max_score,
create_user, create_date, modify_user, modify_date, status.name, type.name
FROM scorecard
JOIN scorecard_type_lu AS type ON
scorecard.scorecard_type_id=type.scorecard_type_id
JOIN scorecard_status_lu AS status ON
scorecard.scorecard_status_id=status.scorecard_status_id
WHERE scorecard_id IN (id1, id2,..,idN);

For each row returned
       Create a new Scorecard instance with the retrieved information
       Create new ScorecardType and ScorecardStatus instances to set to the
Scorecard instance
       Check if the current scorecardId exists in “inUseIds” list. Then set the
scorecard in use property accordingly.
       Store the Scorecard instance in a list (scorecards)
End For

If the “complete” parameter is false, convert the “scorecards” list to array
and return.

Get all group instances for the given scorecard ids.

SELECT scorecard_group_id, scorecard_id, name, weight FROM scorecard_group
WHERE scorecard_id IN (id1, id2,..,idN);

For each row returned
Create a new Group instance with the retrieved information
Store the Group instance in a list (groups)
       Walk the “scorecards” list and match the group with its scorecard using
the retrieved scorecard_id.
       When found the matched scorecard, set the group to the scorecard.
End For
```

```
Get all section instances for the given scorecard ids.

SELECT scorecard_section_id, group.scorecard_group_id,
scorecard.scorecard_id, section.name, section.weight
FROM scorecard_section AS section
JOIN scorecard_group AS group ON section.scorecard_group_id=
group.scorecard_group_id
JOIN scorecard ON group.scorecard_id=scorecard.scorecard_id
WHERE scorecard.scorecard_id IN (id1, id2,..,idN);

For each row returned
Create a new Section instance with the retrieved information
Store the Section instance in a list (sections)
       Walk the “groups” list and match the section with its group using the
retrieved scorecard_group_id.
       When found the matched group, set the section to the group.
End For

Get all question instances for the given scorecard ids.

SELECT question.scorecard_question_id, question.scorecard_question_type_id,
question_type.name, section.scorecard_section_id, group.scorecard_group_id,
scorecard.scorecard_id, question.description, question.guideline,
question.weight, question.upload_document, question.upload_document_required
FROM scorecard_question AS question
JOIN scorecard_question_type_lu AS question_type ON
question.scorecard_question_type_id = question_type.scorecard_question_type_id
JOIN scorecard_section AS section ON
question.scorecard_section_id=section.scorecard_section_id
JOIN scorecard_group AS group ON
section.scorecard_group_id=group.scorecard_group_id
JOIN scorecard ON group.scorecard_id=scorecard.scorecard_id
WHERE scorecard.scorecard_id IN (id1, id2,..,idN);

For each row returned
Create a new Question instance with the retrieved information.
Create a new QuestionType instance and set to the question.
       Walk the “sections” list and match the question with its section using
the retrieved scorecard_section_id.
       When found the matched section, set the question to the section.
End For

Convert the “scorecards” list to array and return.
```

1.3.7 Delete groups/sections/questions

For performance reason, batch deletion is supported for these items. The delete methods receive an array of ids of items to be deleted.

InformixGroupPersistence#deleteGroups() method:

```
Get sectionIds for the given groupIds
       SELECT scorecard_section_id FROM scorecard_section AS section
       JOIN scorecard_group AS group ON
       section.scorecard_group_id=group.scorecard_group_id
       WHERE group.scorecard_group_id IN (grId1, grId2,., grIdN);
Delete the question belong to the obtained sectionIds
       DELETE FROM scorecard_question WHERE scorecard_section_id
       IN (secId1, secId2,., secIdN);
Delete the obtained sections
       DELETE FROM scorecard_section WHERE scorecard_section_id
```

```
       IN (secId1, secId2,.., secIdN);
Delete the groups
       DELETE FROM scorecard_group WHERE scorecard_group_id
       IN (grId1, grId2,.., grIdN);
```

InformixSectionPersistence#deleteSections() method:

```
Delete the question belong to the sectionIds
       DELETE FROM scorecard_question WHERE scorecard_section_id
       IN (secId1, secId2,.., secIdN);
Delete the sections
       DELETE FROM scorecard_section WHERE scorecard_section_id
       IN (secId1, secId2,.., secIdN);
```

InformixSectionPersistence#deleteQuestions() method:

```
DELETE FROM scorecard_question WHERE scorecard_question_id
IN (qId1, qId2,.., qIdN)
```

1.3.8 Initialize the SearchBundle instance and use its for searching

ScorecardManagerImpl#constructor:

```
Load the 'SearchBuilderNamespace' property
Initialize SearchBundleManager using that value
Call manager.getSearchBundle("ScorecardSearchBundle") to get the
SearchBundle(searchBundle)
Create a map with the following key/value pairs, this is required by
SearchBundle (validationMap)
        "ScorecardStatusID"/LongValidator.isPositive()
        "ScorecardTypeID"/LongValidator.isPositive()
        "ProjectCategoryID"/LongValidator.isPositive()
        "ProjectID"/LongValidator.isPositive()
        "ScorecardStatusName"/
StringValidator.hasLength(IntegerValidator.lessThanOrEqualTo(64))
        "ScorecardTypeName"/
StringValidator.hasLength(IntegerValidator.lessThanOrEqualTo(64))
        "ScorecardName"/
StringValidator.hasLength(IntegerValidator.lessThanOrEqualTo(64)));
        "ScorecardVersion"/
StringValidator.hasLength(IntegerValidator.lessThanOrEqualTo(16)));
Call searchBundle.setSearchableFields(validationMap) to set the validation map
```

ScorecardManagerImpl#searchScorecards() method:

Use the SearchBundle instance to search for ids, then the ScorecardPersistence instance to get Scorecard instance array. The ‘searchBundle’ here is the member of ScorecardManagerImpl class. CustomResultSet – class of Database Abstraction component – is used by SearchBuilder to store search result.

.

```
       CustomResultSet result = (CustomResultSet)searchBundle.search(filter);
       Get the scorecardIds from result
       Get Scorecard instances using persistence. getScorecards() method with
       “scorecardIds” and “complete” parameter.
       Return the Scorecard instances.
```

1.4 Component Class Overview

ScorecardManager

This interface defines the contract for scorecard manager. A scorecard manager implementation has the responsibility to validate and create/update/retrieve/search scorecard instances in the persistence. The manager read configuration settings to load the configured persistence and validator implementation.

ScorecardManagerImpl
This is the manager class of this component. It loads persistence implementation using settings in the configuration namespace. Then it uses the persistence implementation to create/update/retrieve/search scorecards. This is the main class of the component, which is used by client to perform the above scorecards operations. The default configuration namespace for this class is: "com.topcoder.management.scorecard". It can accept a custom namespace as well. Apart from the persistence settings, it also initializes a SearchBundle instance to use in scorecards searching and a ScorecardValidator instance to validate scorecards.

ScorecardSearchBundle
This class contains methods to build filter instances to use in scorecard searching. It can build filter to search for scorecard based on various criteria such as: - Scorecard name - Scorecard version - Scorecard type id - Scorecard type name - Scorecard status id - Scorecard status name - Project category id that the scorecard linked to. - Project id that the scorecard linked to. Besides, this class also provides method to combine any of the above filter to make complex filters. This class is used be the client to create search filter. The created filter is used in searchScorecards() method of ScorecardManager.

ScorecardPersistence
This interface defines the contract that any scorecard persistence must implenment. The implementation classes will be used by ScorecardManagerImpl to perform scorecard persistence operations. The implementation classes should have a constructor that receives a namespace string parameter so that they're exchangeable with each other by changing configuration settings in the manager.

InformixScorecardPersistence
This class contains operations to create and update scorecard instances into the Informix database. It implements the ScorecardPersistence interface to provide a plug-in persistence for Informix database. It is used by the ScorecardManagerImpl class. The constructor takes a namespace parameter to initialize database connection. Note that in this class, delete operation is not supported. To delete the scorecard, its status is set to 'Disabled'. The create and update operation here work on the scorecard and including its sub items as well. It means creating/updating a scorecard would involve creating/updating its sub groups/sections and questions. The get scorecard operation including a 'complete' parameter, so the scorecard can be retrieve with or without its sub items.
InformixGroupPersistence
This class contains operations to create and update group instances into the Informix database. It is package level because it is used only in InformixScorecardPersistence class to persist group information. Connection to the database is passed to this class during initialization.

InformixSectionPersistence
This class contains operations to create and update section instances into the Informix database. It is package level because it is used only in InformixGroupPersistence class to persist section information. Connection to the database is passed to this class during initialization.

InformixQuestionPersistence
This class contains operations to create and update question instances into the Informix database. It is package level because it is used only in InformixSectionPersistence class to persist question information. Connection to the database is passed to this class during initialization.

ScorecardValidator
This interface defines the contract that scorecard validators should implement. The implementation classes will be used by ScorecardManagerImpl to perform scorecard validation. ScorecardManagerImpl loads the validation implementation from the configuration settings, which allows further validator to plug-in. The implementation classes should have a constructor that receives a namespace string parameter so that they're exchangeable with each other by changing configuration settings in the manager.

DefaultScorecardValidator
This is the default implementation of the ScorecardValidator interface to provide scorecard validation functions. It validates the scorecard base on the following rules: 1/ For scorecard: - scorecard_status_id: Must be greater than zero - scorecard_type_id: Must be greater than zero - project_category_id: Must be greater than zero - name: Must not empty. Length must smaller than or equal to 64 - version: Must not empty and have at least one dot character. Length smaller than 16 - min_score: Must be greater than or equal to zero - max_score: Must be greater than min_score. 2/ For group: - name: Length must smaller than or equal to 64 - weight: Must be greater than or equal to zero and smaller than or equal to 100 - Total weight must equals to 100 3/ For section: - name: Length must smaller than or equal to 64 - weight: Must be greater than or equal to zero and smaller than or equal to 100 - Total weight must equals to 100 4/ For question: - name: Must not empty. Length must smaller than or equal to 64 - description: Must not be empty - guideline: Can be null or empty - weight: Must be greater than or equal to zero and smaller than or equal to 100 - Total weight must equals to 100

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 classes that have to load configuration settings such as ScorecardManagerImpl and InformixScorecardPersistence.

PersistenceException [Custom]

Represents an exception related to persistence operations such as cannot create connection, database table does not exist, etc. Inner exception should be provided to give more details about the error. It is used in persistence implementation classes.

ValidationException [Custom]

Represents an exception related to validating scorecard. Inner exception should be provided to give more details about the error. It is used in classes of the validation package.

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), arrays with null elements.

1.6 Thread Safety

This design is not thread safe because thread-safety is not required. Two threads running the same method will use the same statement and could overwrite each other's work. To achieve thread-safety in a multi-threading environment, data classes such as Scorecard, Group, Section, and Question should not be shared among different threads. That mean each thread should create its own data classes to persist scorecard data. ScorecardSearchBundle and DefaultScorecardValidator classes are immutable so they're thread-safe by default.

2. Environment Requirements

2.1 Environment

JDK 1.3

2.2 TopCoder Software Components

Configuration Manager v2.1.3 – used to read the configuration information.
DB Connection Factory v1.0 – used to create the connection to the database.
Search Builder v1.3 – used to provide scorecard search functions.
ID Generator v3.0 – used to generate ids for scorecards, groups, sections and questions.
Base Exception 1.0 – used as the base for all custom exception classes.
Data Validation 1.0 – used to validate scorecards.

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.topcoder.management.scorecard
com.topcoder.management.scorecard.persistence
com.topcoder.management.scorecard.validation

3.2 Configuration Parameters

For ScorecardManagerImpl class:

Parameter	Description	More information
SearchBuilderNamespace	The namespace that contains settings for SearchBuilder.	String – Required

PersistenceClass	The full class name of the persistence implementation.	String – Required
PersistenceNamespace	The namespace that contains setting for the persistence implementation. If missing, value of 'PersistenceClass' will be used.	String - Optional
ValidatorClass	The full class name of the validation implementation.	String – Required
ValidatorNamespace	The namespace that contains setting for the validation implementation. If missing, value of 'ValidatorClass' will be used.	String - Optional

For InformixScorecardPersistence class:

Parameter	**Description**	**More information**
ConnectionFactoryNS	The namespace that contains settings for DB Connection Factory.	String – Required
ConnectionName	The name of the connection that will be used by DBConnectionFactory to create connection. If missing, default connection will be created.	String - Optional

Search Builder alias: These alias setting affect the logic of the component. Therefore, they should not be changed. They are defined as public constant in ScorecardSearchBundle class and used in the component logic.

Alias	**Description**	**Value**
ScorecardStatusID	Alias for scorecard status id column.	scorecard_status_lu.scorecard_status_id
ScorecardStatusName	Alias for scorecard status name column.	scorecard_status_lu.name
ScorecardTypeID	Alias for scorecard type id column.	scorecard_type_lu.scorecard_type_id
ScorecardTypeName	Alias for scorecard type name column.	scorecard_type_lu.name
ProjectCategoryID	Alias for project category id column.	project_category_lu.project_category_id
ProjectID	Alias for project id column.	project_scorecard.project_id
ScorecardName	Alias for scorecard name column.	scorecard.name
ScorecardVersion	Alias for scorecard verssion column.	scorecard.version

*This is SQL command used as 'context' property in SearchBuilder setting. Note the use of **LEFT JOIN***
*on project_scorecard table. This is make sure scorecard id is still retrieved even it does not exist in project_scorecard table. **It ensures both in-use and not in-use scorecard are returned from the search.***

```
SELECT scorecard.scorecard_id FROM scorecard
JOIN scorecard_status_lu ON
scorecard.scorecard_status_id=scorecard_status_lu.scorecard_status_id
JOIN scorecard_type_lu ON
scorecard.scorecard_type_id=scorecard_type_lu.scorecard_type_id
JOIN project_category_lu ON
scorecard.project_category_id=project_category_lu.project_category_id
LEFT JOIN project_scorecard ON
scorecard.scorecard_id=project_scorecard.scorecard_id WHERE
```

Sample configuration file:

```
<?xml version="1.0" ?>
<CMConfig>
    <!-- Namespace for ScorecardManagerImpl class -->
    <Config name="com.topcoder.management.scorecard">
        <Property name="SearchBuilderNamespace">
            <Value>com.topcoder.searchbuilder</Value>
        </Property>

        <Property name="PersistenceClass">

<Value>com.topcoder.management.scorecard.persistence.InformixScorecardPersisten
ce</Value>
        </Property>

        <Property name="PersistenceNamespace">

<Value>com.topcoder.management.scorecard.persistence.InformixScorecardPersisten
ce</Value>
        </Property>

        <Property name="ValidatorClass">

<Value>com.topcoder.management.scorecard.validation.DefaultScorecardValidator</
Value>
```

```
        </Property>

        <Property name="ValidatorNamespace">

<Value>com.topcoder.management.scorecard.validation.DefaultScorecardValidator</
Value>
        </Property>
    </Config>

    <!-- Namespace for InformixScorecardPersistence class -->
    <Config
name="com.topcoder.management.scorecard.persistence.InformixScorecardPersistenc
e">
        <Property name="ConnectionFactoryNS">
            <Value>Dbconnection.factory</Value>
        </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</Valu
e>
                </Property>

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

                    <Property name="jdbc_url">
                        <Value>jdbc:informix-sqli://zeyuanzhu-
home:1526/score:INFORMIXSERVER=ol_home</Value>
                    </Property>

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

                    <Property name="DatabaseName">
                        <Value>score</Value>
                    </Property>

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

                    <Property name="password">
                        <Value>12345</Value>
                    </Property>
                </Property>
            </Property>
        </Property>
    </Config>

  <!-- Namespace for SearchBuilder component
```

```
    The setting in this section decide the logic of secorecard searching.
   -->
  <Config name="com.topcoder.searchbuilder">
        <Property name="searchBundles">
            <Property name="ScorecardSearchBundle">
                      <Property name="type">
                    <Value>Database</Value>
              </Property>
                      <Property name="name">
                    <Value>ScorecardSearchBundle</Value>
              </Property>
                      <Property name="context">
                       <Value>SELECT scorecard.scorecard_id FROM scorecard
JOIN scorecard_status_lu ON
scorecard.scorecard_status_id=scorecard_status_lu.scorecard_status_id
JOIN scorecard_type_lu ON
scorecard.scorecard_type_id=scorecard_type_lu.scorecard_type_id
JOIN project_category_lu ON
scorecard.project_category_id=project_category_lu.project_category_id
LEFT JOIN project_scorecard ON
scorecard.scorecard_id=project_scorecard.scorecard_id WHERE</Value>
                </Property>

            <Property name="DBNamcespace">
                    <Value>Dbconnection.factory</Value>
            </Property>
            <Property name="connectionProducerName">
                    <Value>dbconnection</Value>
            </Property>

            <Property name="alias">
                <Property name="ScorecardStatusID">
                    <Value>scorecard_status_lu.scorecard_status_id</Value>
                 </Property>
                 <Property name="ScorecardStatusName">
                    <Value>scorecard_status_lu.name</Value>
                 </Property>
                <Property name="ScorecardTypeID">
                    <Value>scorecard_type_lu.scorecard_type_id</Value>
                </Property>
                <Property name="ScorecardTypeName">
                    <Value>scorecard_type_lu.name</Value>
                </Property>
                <Property name="ProjectCategoryID">
                    <Value>project_category_lu.project_category_id</Value>
                </Property>
                <Property name="ProjectID">
                    <Value>project_scorecard.project_id</Value>
                </Property>
                <Property name="ScorecardName">
                    <Value>scorecard.name</Value>
                </Property>
                <Property name="ScorecardVersion">
                    <Value>scorecard.version</Value>
                </Property>

            </Property>
        </Property>
       </Property>
  </Config>
</CMConfig>
```

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. Follow the demo.

4.3 Demo

4.3.1 General usage demo

```
// create scorecard manager with default ns
ScorecardManager manager = new ScoreCardManagerImpl();

// create a scorecard
Scorecard scorecard = ...;

// insert scorecard into the persistence, the operator is "user1"
manager.createScorecard(scorecard, "user1");

// the scorecard id is updated
System.out.println("New scorecard id: " + scorecard.getId());

// the scorecard creation user is updated
System.out.println("Creation user: " + scorecard.getCreationUser());

// makesome modification to the scorecard
scorecard.addGroup(...);

// update scorecard in the persistence, the operator is "user2"
manager.updateScorecard(scorecard, "user2");

// the scorecard modification user is updated
System.out.println("Modification user: " +
scorecard.getModificationUser());

// retrieve a scorecard
Scorecard newScorecard = manager.getScorecard(scorecard.getId());

// build a filter for searching scorecard, this case is the filter for
type id equals 1
Filter f1 = ScorecardSearchBundle.buildTypeIdEqualFilter(1);

// light weight searching, which only retrieve scorecards without theirs
sub items such as
// groups, sections, questions
Scorecard[] lightResult = manager.searchScorecards(f1, false);

// complete searching, which retrieve the scorecards and their sub items
Scorecard[] completeResult = manager.searchScorecards(f1, true);

// retrieve all scorecard types
```

```
        ScorecardType[] scorecardTypes = manager.getAllScorecardTypes();

        // retrieve all scorecard statuses
        ScorecardStatus[] scorecardStatuses = manager.getAllScorecardStatuses();

        // retrieve all question types
        QuestionType[] questionTypes = manager.getAllQuestionTypes();
```

4.3.2 Build filters demo:

```
    // get filter for scorecard with type id equals 1
        Filter f1 = ScorecardSearchBundle.buildTypeIdEqualFilter(1);

        // get filter for scorecard with type id equals 1, 3, or 4
        List typeIds = new ArrayList();
        typeIds.add(new Long(1));
        typeIds.add(new Long(3));
        typeIds.add(new Long(4));
        Filter f2 = ScorecardSearchBundle.buildTypeIdInFilter(typeIds);

        // get filter for scorecard with type name equals 'Design'
        Filter f3 = ScorecardSearchBundle.buildTypeNameEqualFilter("Design");

        // get filter for scorecard with type name equals 'Design' or
'Development'
        List typeNames = new ArrayList();
        typeNames.add("Design");
        typeNames.add("Development");
        Filter f4 = ScorecardSearchBundle.buildTypeNameInFilter(typeNames);

        // get filter for scorecard with status id equals 1
        Filter f5 = ScorecardSearchBundle.buildStatusIdEqualFilter(1);

        // get filter for scorecard with project category id equals 1
        Filter f6 = ScorecardSearchBundle.buildProjectCategoryIdEqualFilter(1);

        // get filter for scorecard with project id equals 1
        Filter f7 = ScorecardSearchBundle.buildProjectIdEqualFilter(1);

        // get filter for scorecard with name equals 'Test'
        Filter f8 = ScorecardSearchBundle.buildNameEqualFilter("Test");

        // get filter for scorecard with version equals '1.0'
        Filter f9 = ScorecardSearchBundle.buildNameEqualFilter("1.0");
```

5. Future Enhancements

Additional databases could be supported.

Additional validator could be supported to validate scorecard.