Project Management 1.1 Component Specification

Aspects of the design that are changed in version 1.1 are blue.

Additions in version 1.1 are red.

The component provides project management functionalities. Each project belongs to a project category, and has a status. Application can use the component to create, update and search projects. The project persistence logic is pluggable.

The component provides the management functionalities to create/update/retrieve and search projects. The project persistence logic is pluggable.

The main class of this component is ProjectManagerImpl, which implements ProjectManager interface. This class loads persistence and validation implementation from a configuration namespace. The persistence implementation is used to create/update/retrieve projects. The validation implementation is used to validate project instance before persisting. The manager also contains logic to initialize and use SearchBuilder component to search for projects base on various search conditions. Apart from project management operations, the component also provides methods to retrieve values from project related lookup tables.

The ProjectPersistence defines contract for persistence implementations. It allows other implementation to plug-in in the future. For this version, Informix database implementation is provided.
The manager loads persistence implementation using settings from a configuration namespace.

By using SearchBuilder component, various type of search condition can be created to search for projects. This component provides some convenient methods to create search filter. Current search filter includes: Project type/ category/status/project's extended property/project resource's extended property. The search filters can be combined using AND/OR/NOT.

The ProjectValidator interface defines the contract for project 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 project to make sure it conforms to the database schema. Validator settings are also configurable in the ProjectManagerImpl class.

The design provides some enhancement features such as:

- Additional look up operations such as: Get all project types; Get all project property type. Project property type is a list of defined property name. Only property names belong to this list can be set to a Project.
- Project resource property search: Each project can be associated with some resources. Each resource may contain extended properties. This search based on the name and value of the properties of resource to search for projects.
- The select command used in SearchBuilder for searching projects use table join instead of nested query, which will improve performance significantly.
- Pluggable validation implementation.

Note: The main package and persistence implementation will be separated into two development projects. The persistence implementation project contains classes in "persistence" package. The rest belong to the main package project.

Version 1.1 provides two DAO implementations, one that manages its own transactions and is fully backward compatible, and one that relies on externally managed transactions. Both DAO implementations are identical in all respects other than transaction management.

This design realizes the requirements by refactoring most of the logic into an abstract superclass of both concrete DAOs. The template method pattern is used, with the abstract class performing the bulk of the logic for each method, but relying on the subclasses to open and close database connections. If transaction management is needed, it is incorporated into the implementations of these methods. Because of this design, none of the main logic of the component is duplicated, and the logic to open and close connections and the logic to commit transactions is only implemented a single time for each concrete class.

New Version 1.1:
To manage review application for a new project, a table has been added. This table is called review_applications and establishes an m:n relation between resources (reviewers) and project ids. An entry into this table does not mean that the resource (in this case reviewer) is actually assigned to the project, it only means that the reviewer will be considered in the assignment/selection algorithm. All necessary updated and new methods to manage the review_applications table are added into ApplicationsManager interface and corresponding persistence interface.

A new column review_system_version is added into project_info_type_lu table, and a new field reviewSystemVersion is added to ProjectType class. AbstractInformixProjectPersistence is also updated.

New functionalities are added to manage ReviewApplication entities, CRUD operations are provided.

Project review payment and eligibility
The review payment is needed in a few places in the new review system. These properties will be made available through the Project.properties:Map. The persistence templates populating default projects will be modified to store this value in the entity. The following keys will be used:

- "primaryReviewPayment"
- "secondaryReviewPaymentPool"
- "eligibilityPointsPool"

These keys should be added into project_info table as lookup values. Please refer to CS 1.3.2, where we can see all pre-defines keys can be retrieved from project_info table.

So, we create new Project objects by using ProjectManager, the Project object should be configured properly, above keys in Project.properties should be set.

1.1 Design Patterns

Strategy pattern is used in ProjectValidator and ProjectPersistence interface. This allows plugging in different implementations of project validator and persistence. ApplicationsManagerImpl also uses strategy pattern, it holds an instance of ReviewApplicationPersistence that can be different implementations.

Facade pattern is used in ProjectManagerImpl class to provide access to the component functions. This class utilizes validator and persistence implementations.

Template method pattern is used by AbstractInformixProjectPersistence and its subclasses. The main logic for persistence operations is in AbstractInformixProjectPersistence, but the logic of creating, opening, and closing connections and any necessary transaction management is handled by concrete implementations of the protected abstract methods openConnection(), closeConnection(), and closeConnectionOnError().

AbstractInformixReviewApplicationPersistence also uses Template method pattern. The main logic for persistence operations is in AbstractInformixReviewApplicationPersistence, but the logic of creating, opening, and closing connections and any necessary transaction management is handled by concrete implementations of the protected abstract methods openConnection(), closeConnection(), and closeConnectionOnError().

1.2 Industry Standards

JDBC, XML, SQL

1.3 Required Algorithms

1.3.1 Transaction management

Transactions are managed in the concrete subclasses of AbstractInformixProjectPersistence, which are InformixProjectPersistence and UnmanagedTransactionInformixProjectPersistence. InformixProjectPersistence handles its own transactions while UnmanagedTransactionInformixProjectPersistence relies on externally managed transactions.

Connections are opened using the openConnection() method. When this method is called on InformixProjectPersistence, a connection will be opened using the connectionFactory and (if not null) connectionName. The retrieved Connection will be opened, and setAutoCommit(false) will be called on it before returning. UnmanagedTransactionInformixProjectPersistence will follow the same logic, except that setAutoCommit() won't be called.

In the closeConnection() method, InformixProjectPersistence will commit the transaction on the Connection before closing it, by calling the commit() method on the Connection. UnmanagedTransactionInformixProjectPersistence will simply call close() on the given Connection without first committing anything.

If an error occurs while preparing or executing PreparedStatements or processing ResultSets, the connection will need to be disposed of before the PersistenceException is thrown. If there is any transaction, it should be rolled back rather than committed, so a separate method is provided for this case - closeConnectionOnError(). The implementation of this method for UnmanagedTransactionInformixProjectPersistence should be identical to that class's implementation of closeConnection(). InformixProjectPersistence will need to call rollback() instead of commit() on the Connection.

1.3.2 Get project property from persistence using a project_id

```
   Get all defined project property type using method
getAllProjectPropertyTypes() (propertyTypes)

   Get property values for the given project:

SELECT project_info_type_id, value FROM project_info WHERE project_id=?

   Use the project_info_type_id to look up property names from propertyTypes
array. Store the name/value pairs to a map
```

1.3.3 Create project and its associated items

```
   Use IDGenerator to generate new Id for project

   Create project data to its table
INSERT INTO project(project_id, project_status_id, project_category_id,
create_user, create_date, modify_user, modify_date) VALUES
(?,?,?,?,CURRENT,?,CURRENT)

   Create project properties

      Get all defined project property types using method
      getAllProjectPropertyTypes() (propertyTypes)

      For each property in properties map, use property name to look up for
      project_info_type_id from propertyTypes array.

INSERT INTO project_info(project_id, project_info_type_id, value, create_user,
create_date, modify_user, modify_date) VALUES (?,?,?,?,CURRENT,?,CURRENT)

      End for
```

```
If everything is fine, set these value back to the project instance
   - Generated project id.
   - Creation user/Creation date
   - Modification user/Modification date
```

1.3.4 *Update project and its associated items*

Throw exception if the project with given id does not exist in the persistence.

Gets all the properties from project_info table which belong to the given project. Assume that propertyIdSet keeps the property ids retrieved in this step.

```
    Update project data to its table
UPDATE project SET project_status_id=?, project_category_id=?, modify_user=?,
modify_date=CURRENT WHERE project_id=?

    Update project properties
       Get all defined project property types using method
       getAllProjectPropertyTypes() (propertyTypes)

       Compare the properties from “oldProject” (using propertyIdSet) with current
       property from the instance to find out: Added properties/Removed
       properties.

       Use property name to look up for project_info_type_id from propertyTypes
       array (for all add/remove/update actions). If lookup failed throw
       PersistenceException.

       For each new properties
INSERT INTO project_info(project_id, project_info_type_id, value, create_user,
create_date, modify_user, modify_date)
VALUES (?,?,?,?,CURRENT,?,CURRENT)
       End for

       With the removed properties, batch deleting them
DELETE FROM project_info WHERE project_id=? AND project_info_type_id IN (type1,
type2,..,typeN)

       For each of the remaining properties
UPDATE project_info SET value=?, modify_user=?, modify_date=CURRENT WHERE
project_id=? AND project_info_type_id=?
       End for

    Add update reason into project_audit table:
       Generate new id for “project_audit_id” column.
       Creation and modification user for this table is the operator.
INSERT INTO project_audit(project_audit_id, project_id, update_reason,
create_user, create_date, modify_user, modify_date)
VALUES (?,?,?,?,CURRENT,?,CURRENT)

    If everything is fine, set these value back to the project instance
       - Modification user/Modification date
```

1.3.5 *Get project using project_id*

Reuse the logic in “Get an array of projects using project_ids” with an array contains only one id.

1.3.6 *Get an array of projects using project_ids*

This method gets an array of project instances from the database using minimum SQL commands to improve performance.

Get project information for the given project ids

```
SELECT project.project_id,
project.project_status_id, status.name,
project.project_category_id, category.name,
category.project_type_id, type.name,
project.create_user, project.create_date,
project.modify_user, project.modify_date,
FROM project
JOIN project_status_lu AS status
ON project.project_status_id = status.project_status_id
JOIN project_category_lu AS category
ON project.project_category_id = category.project_category_id
JOIN project_type_lu AS type
ON category.project_type_id = type.project_type_id
WHERE project_id IN (id1, id2, ..., idN)
```

With each returned row, create a Project instance with its category and status.

The result is an array of project instance (projects[])

Get project properties for the given project ids

```
SELECT project.project_id, info_type.name, info.value
FROM project
JOIN project_info AS info
ON project.project_id = info.project_id
JOIN project_info_type_lu as info_type
ON info.project_info_type_id = info_type.project_info_type_id
WHERE project.project_id IN (id1, id2, ..., idN)
```

For each returned row, match the project property with its project using project_id and then set the property to its project.

1.3.7 *Search for projects*

Initialize SearchBuilder component in ProjectManagerImpl#constructor:
Search builder requires that validator must be set for all search alias.

```
Load the 'SearchBuilderNamespace' property
Initialize SearchBundleManager using that value
Call manager.getSearchBundle("ProjectSearchBundle") to get the
SearchBundle(searchBundle)
Create a map with the following key/value pairs, this is required by
SearchBundle (validationMap)
    "ProjectTypeID"/IntegerValidator.isPositive()
    "ProjectCategoryID"/IntegerValidator.isPositive()
    "ProjectStatusID"/IntegerValidator.isPositive()
    "ProjectTypeName"/
StringValidator.hasLength(IntegerValidator.lessThan(64))
    "ProjectCategoryName"/
StringValidator.hasLength(IntegerValidator.lessThan(64))
    "ProjectStatusName"/
StringValidator.hasLength(IntegerValidator.lessThan(64)));
    "ProjectPropertyName"/
StringValidator.hasLength(IntegerValidator.lessThan(64)));
```

```
    "ProjectPropertyValue"/
StringValidator.hasLength(IntegerValidator.lessThan(64)));
    "ProjectResourcePropertyName"/
StringValidator.hasLength(IntegerValidator.lessThan(64)));
    "ProjectResourcePropertyValue"/
StringValidator.hasLength(IntegerValidator.lessThan(64)));

Call searchBundle.setSearchableFields(validationMap) to set the validation
map.
```

Search logic in ProjectManagerImpl#searchProjects(Filter filter) method:
The 'searchBundle' is the member of ProjectManagerImpl.

```
CustomResultSet result = (CustomResultSet) searchBundle.search(filter);
List ids = new ArrayList();
while (result.next()) {
    long id = result.getLong(0);
    ids.add(new Long(id));
}

// convert the ids list to an array (idArray)
return persistence.getProjects(idArray)
```

1.3.8 Search for active project associated with an external user id

Each project can associate with zero or more resources. Each resource can have zero or more properties. Searching for project associated with an external user id actually is searching for project that contains resource with the resource property name "External Reference ID". The value of this property is the user id in string format. Active projects are projects with status set to "Active". Since the search function of this component support search for project using resource property name/value pair and status name, perform this task become simple.
Here is the code for ProjectManagerImpl#getUserProjects(long user) method:

```
Filter resourcePropFilter =
ProjectFilterUtility.buildProjectPropertyResourceEqualFilter("External Reference ID",
String.valueOf(user));
        Filter activeStatusFilter =
ProjectFilterUtility.buildStatusNameEqualFilter("Active");
        return searchProjects(ProjectFilterUtility.buildAndFilter(resourcePropFilter,
activeStatusFilter));
```

1.3.9 Create ReviewApplication

```
Execute following SQL statement:
insert into review_applications (id, reviewer_id, project_id, application_date,
is_primary) values (?,?,?,?,?)
1st parameter: use reviewApplicationIDGenerator to generate new ID.
2nd parameter: reviewerApplication.reviewerId
3rd parameter: reviewerApplication.projectId
4th parameter: reviewerApplication.applicationDate
5th parameter: reviewerApplication.isPrimary

Set generated ID to reviewApplication.

To make the content of cache consistent with data in database, we have to remove
some old data. For example, a new ReviewApplication is added into database with
projected=1, then the cached data for projectId=1 is out of date, it should be
removed from cache, when data for that project is requested again, the updated
data will be retrieved from database and stored in cache.

Remove the entry in cache with key=PRIMARY_PRIFIX+projectId
Remove the entry in cache with key=SECONDARY_PRIFIX+projectId
Remove the entry in cache with key=ALL_PREFIX+projectId
```

Put reviewApplication into cache with key=SINGLE_PREFIX+id.

Return reviewApplication.

1.3.10 Update ReviewApplication

Check if review application data exists in persistence:
select id from review_applications where id=?
If result set is empty, throw ReviewApplicationPersistenceException.

Update review application data by executing:
update review_applications set reviewer_id=?, project_id=?, application_date=?, is_primary=? where id=?

Remove the entry in cache with key=PRIMARY_PRIFIX+projectId
Remove the entry in cache with key=SECONDARY_PRIFIX+projectId
Remove the entry in cache with key=ALL_PREFIX+projectId

Put ReviewApplication entity into cache with key=SINGLE_PREFIX+id.

Return the updated ReviewApplication

1.3.11 Retrieve ReviewApplication By ID

Check if there is any entry in cache with key=SINGLE_PREFIX+id. If yes, return the cached object directly.

Get review application entity from database by executing:
select review_id, project_id, application_date, is_primary from review_applications where id=?

If result set is empty, return null; Else populate a ReviewApplication entity with data in result set.

Put ReviewApplication entity into cache with KEY=SINGLE_PREFIX+id

Return ReviewApplication entity.

1.3.12 Delete ReviewApplication by ID

Get entity project id by executing:
select project_id from review_applications where id=?

If result set is null, return false.

Remove the entry in cache with key=PRIMARY_PRIFIX+projectId
Remove the entry in cache with key=SECONDARY_PRIFIX+projectId
Remove the entry in cache with key=ALL_PREFIX+projectId

Delete entity from persistence by executing:
delete from review_applications where id=?
Return true;

1.3.13 Get primary applications

Check whether there is an object in cache with key=PRIMARY_PRIFIX+projectId, if yes, return the object directly.

Retrieve applications from persistence by executing:
select id, reviewer_id, application_date from review_applications where project_id=? and is_primary='t'

If result set is empty, return empty array; Else populate a ReviewApplication array with data in result set.

Put the array into cache with key=PRIMARY_PREFIX+projectId

```
Return the array.
```

1.3.14 *Get the secondary applications*

```
Check whether there is an object in cache with key=SECONDARY_PRIFIX+projectId, if
yes, return the object directly.

Retrieve applications from persistence by executing:
select id, reviewer_id, application_date from review_applications where
project_id=? and is_primary='f'

If result set is empty, return empty array; Else populate a ReviewApplication
array with data in result set.

Put the array into cache with key=SECONDARY_PREFIX+projectId

Return the array.
```

1.3.15 *Get all applications*

```
Check whether there is an object in cache with key=ALL_PRIFIX+projectId, if yes,
return the object directly.

Retrieve applications from persistence by executing:
select id, reviewer_id, application_date, is_primary from review_applications
where project_id=?

If result set is empty, return empty array; Else populate a ReviewApplication
array with data in result set.

Put the array into cache with key=ALL_PREFIX+projectId

Return the array.
```

1.4 Component Class Overview

ProjectManager
This interface defines the contract for project manager. A project manager implementation has the responsibility to validate and create/update/retrieve/search project instances in the persistence. The manager read configuration settings to load the configured persistence and validator implementation.
The manager also provides methods to load an array of project associated with an external user id, get all project categories, and project statuses from the persistence.

ProjectManagerImpl
This is the manager class of this component. It loads persistence implementation using settings in the configuration namespace. Then use the persistence implementation to create/update/retrieve/search projects. This is the main class of the component, which is used by client to perform the above project operations. Searching projects and getting project associated with and external user id are two operations which the logic reside in this class. The remaining operations are delegated to persistence implementation.
The default configuration namespace for this class is: "com.topcoder.management.project".

Project
This class represents a project from the persistence. Each project must belong to a project category and must have a status. Project also contains some defined propeties. Projects are stored in 'project' table, project category in 'project_category_lu' table, project status in 'project_status_lu' table. Project properties are stored in 'project_info' table. This class is used by ProjectPersistence implementors to store projects in the persistence.
This class implements Serializable interface to support serialization.

ProjectType
This class represents a project type from the persistence. Each project category must belong to a project type. Project types are stored in 'project_type_lu' table, project category in 'project_category_lu'. A project

type instance contains id, name and description. This class is used in ProjectCategory class to specify the project type of the project category. This class implements Serializable interface to support serialization.

ProjectCategory
This class represents a project category from the persistence. Each project category must belong to a project type. Project type are stored in 'project_type_lu' table, project category in 'project_category_lu'. A project category instance contains id, name and description and a reference to project type. This class is used in Project class to specify the project category of a project. This class implements Serializable interface to support serialization.

ProjectStatus
This class represents a project status from the persistence. Project statuses are stored in 'project_status_lu' table. A project status instance contains id, name and description. This class is used in Project class to specify the project status of a project. This class implements Serializable interface to support serialization.

ProjectPropertyType
This class represents a project property type from the persistence. Each project property must associated with a project property type. Project property types are stored in 'project_info_type_lu' table, project property in 'project_info' table. A project property type instance contains id, name and description. This class is used in ProjectManager#getAllProjectPropertyTypes method to return project property types from the persistence. This class implements Serializable interface to support serialization.

ProjectFilterUtility
This class contains methods to build filter instances to use in project searching. It can build filter to search for project based on various criteria such as:

- Project type id
- Project type name
- Project category id
- Project category name
- Project status id
- Project status name
- Project property name
- Project property value
- Project resource property name
- Project resource property value

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 SearchProjects() method of ProjectManager.

ProjectPersistence
This interface defines the contract that any project persistence must implement. The implementation classes will be used by ProjectManagerImpl to perform project 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.

AbstractInformixProjectPersistence
The AbstractInformixProjectPersistence class implements the ProjectPersistence interface, in order to persist to the database structure in the project_management.sql script. It contains most of the logic that was in the InformixProjectPersistence class in version 1.0.1.

This class does not cache a Connection to the database. Instead, it uses the concrete implementation of the openConnection() method of whatever subclass is in use to acquire and open the Connection. After the queries are executed and the result sets processed, it uses the closeConnection() method to dispose of the connection. If the operation fails, closeConnectionOnError() is called instead. This allows the transaction handling logic to be implemented in subclasses while the statements, queries, and ResultSets are handled in the abstract class.

Note that in this class, delete operation is not supported. To delete a project, its status is set to 'Deleted'. Create and update operations work on the project and its related items as well. This means creating/updating a project involves creating/updating its properties.

This class is immutable and thread-safe.

InformixProjectPersistence
The InformixProjectPersistence class in version 1.1 is completely compatible with the InformixProjectPersistence class in version 1.0.1 and has no additional functionality. However, the bulk of the logic that was in this class in version 1.0.1 has been moved into an abstract superclass that is the base for UnmanagedTransactionInformixProjectPersistence as well. The only logic remaining in this class is that of opening connections and managing transactions, and the only methods implemented in this class are openConnection(), closeConnection(), and closeConnectionOnError(), which are concrete implementations of the corresponding protected abstract methods in AbstractResourcePersistence and are used in the context of a Template Method pattern.

This class is immutable and thread safe.

UnmanagedTransactionInformixProjectPersistence
The UnmanagedTransactionInformixProjectPersistence class is a new class in version 1.1. It performs exactly the same tasks as InformixProjectPersistence, but is designed to be used with externally managed transactions. The implementations of openConnection(), closeConnection(), and closeConnectionOnError() in this class do not call commit(), setAutoCommit(), or rollback(), as the transaction is expected to be handled externally to the component.

This class is immutable and thread-safe.

ProjectValidator
This interface defines the contract that project validators should implement. The implementation classes will be used by ProjectManagerImpl to perform project validation. ProjectManagerImpl 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. The set of rules is the logic of implementation classes.

DefaultProjectValidator
This is the default implementation of the ProjectValidator interface to provide project validation functions.

- It validates the project base on the following rules:
- Project type/status/category name: Length must be less than 64
- Project type/status/category description: Length must be less than 256
- Project property key: Length must be less than 64
- Project property value: Length must be less than 4096

ReviewApplication:
This is the entity class the represents a review application. It holds the association between reviewer and the project and type of the review (primary or secondary).

ApplicationsManager(interface):
This interface defines the contract for managing review applications. It provides CRUD operations for ReviewApplication data.

ReviewApplicationPersistence(interface):
This interface defines the contract for managing ReviewApplication entities in persistence.

ApplicationsManagerImpl:
This class is the default implementation of ApplicationsManager interface. It holds and instance of ReviewApplicationPersistence and delegates all job to ReviewApplicationPersistence instance. It provides CRUD operations of ReviewApplication entity.

AbstractInformixReviewApplicationPersistence(abstract)
This class is the base class for informix review application persistence implementations. It implements CRUD operations on ReviewApplication entities. DBConnectionFactory is used to create database connections. Abstract methods openConnection, closeConnection and closeConnectionOnError are provided to allow child class to manage transaction by itself or not.

InformixReviewApplicationPersistence:
This is a review application persistence implementation. It manages transactions by itself. The only logic remaining in this class is that of opening connections and managing transactions, and the only methods implemented in this class are openConnection(), closeConnection(), and closeConnectionOnError(), which are concrete implementations of the corresponding protected abstract methods in AbstractResourcePersistence and are used in the context of a Template Method pattern.

UnmanagedTransactionInformixReviewApplicationPersistence:
This class performs exactly the same tasks as InformixReviewApplicationPersistence, but is designed to be used with externally managed transactions. The implementations of openConnection(), closeConnection(), and closeConnectionOnError() in this class do not call commit(), setAutoCommit(), or rollback(), as the transaction is expected to be handled externally to the component.

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 ProjectManagerImpl, InformixProjectPersistence and UnmanagedTransactionInformixProjectPersistence.

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 projects. When a project validation failed due to some reason, this exception will be thrown including the validation message.Inner exception should be provided whenever possible 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), and arrays with null elements.

ApplicationsManagerException:
This exception is thrown when any error occurred during managing ReviewApplication entities. It's thrown by ApplicationsManager implementations.

ReviewApplicationPersistenceException:
This exception is thrown when any error occurred during persistence operations of ReviewerStatistics entities. It's thrown by ReviewerStatisticsPersistence implementations.

ReviewApplicationConfigurationException:
This exception is thrown when any error occurred during configuration. It's thrown by ApplicationsManagerImpl and AbstractInformixReviewApplicationPersistence.

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, synchronization the database accessing method of ProjectManagerImpl is needed. ProjectFilterUtility and DefaultProjectValidator are immutable so they are thread-safe by default.

V1.1 doesn't affect the thread safety of this component. As this component is in previous version is not thread safe. ApplicationsManager in this version does not required to be thread safe too, so it doesn't change the thread safety of this component.

2. Environment Requirements

2.1 Environment

JDK 1.4

2.2 TopCoder Software Components

- Configuration Manager v2.1.5 – used to read the configuration information.
- DB Connection Factory v1.0 – used to create the connection to the database.
- Search Builder v1.3.1 – used to provide project search functions.
- ID Generator v3.0 – used to generate ids for project and project audit.
- Base Exception 1.0 – used as the base for all custom exception classes.
- Project Management 1.0 – The project management component.
  Simple Cache 2.0 – Provides caching functionality for this component.

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.project
com.topcoder.management.project.persistence
com.topcoder.management.project.validation

3.2 Configuration Parameters

For ProjectManagerImpl 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 InformixProjectPersistence or UnmanagedTransactionInformixProjectPersistence 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
ProjectIdSequenceName	The sequence name used to create Id generator for project Id. If missing, default value (project_id_seq) is used.	String - Optional
ProjectAuditIdSequenceName	The sequence name used to create Id generator for project audit Id. If missing, default value (project_audit_id_seq) is used.	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 ***ProjectFilterUtility*** *class and used in the component logic.*

Alias	Description	Value
ProjectTypeID	Alias for project type id column.	project_type_lu.project_type_id
ProjectTypeName	Alias for project type name column.	project_type_lu.name
ProjectCategoryID	Alias for project category id column.	project_category_lu.project_category_id
ProjectCategoryName	Alias for project category name column.	project_category_lu.name
ProjectStatusID	Alias for project status id column.	project_status_lu.project_type_id
ProjectStatusName	Alias for project status name column.	project_status_lu.name
ProjectPropertyName	Alias for project property name column.	project_info_type_lu.name
ProjectPropertyValue	Alias for project property value column.	project_info.value
ProjectResourcePropertyName	Alias for project resource property name column.	resource_info_type_lu.name
ProjectResourcePropertyValue	Alias for project resource property value column.	resource_info.value

ApplicationsManagerImpl:

Parameter	Description	Value
PersistenceClassName	Class name of persistence implementation	String, not null or empty
PersistenceNamespace	Configuration namespace for persistence implementation	String, not null or empty

AbstractInformixReviewApplicationPersistence:

Parameter	Description	Value
DBConnFactoryNamespace	Configuration namespace for DB connection factory.	String, not null or empty
ConnectionName	Name of DB connection	String, not null or empty
ReviewApplicationIDSequenceName	ID generator name for ReviewApplication	String, not null or empty
CacheNamespace	Configuration namespace for Cache	String, not null or empty

This is SQL command used as ‘context’ property in SearchBuilder setting. Note the use of **LEFT JOIN** on **project_info, resource** and **resource_info** table. It makes sure the search result also includes projects that do not have properties.

```
SELECT project.project_id FROM project
JOIN project_category_lu ON
project.project_category_id = project_category_lu.project_category_id
JOIN project_status_lu ON
project.project_status_id = project_status_lu.project_status_id
JOIN project_type_lu ON
project_category_lu.project_type_id = project_type_lu.project_type_id
LEFT JOIN project_info ON
project.project_id = project_info.project_id
JOIN project_info_type_lu ON
project_info.project_info_type_id = project_info_type_lu.project_info_type_id
LEFT JOIN resource ON
project.project_id=resource.project_id
LEFT JOIN resource_info ON
resource.resource_id = resource_info.resource_id
```

```
JOIN resource_info_type_lu ON
resource_info.resource_info_type_id =
resource_info_type_lu.resource_info_type_id
```

Sample configuration file:

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

        <Property name="PersistenceClass">
            <Value>com.topcoder.management.project.persistence.InformixProjectPersistence</Value>
        </Property>

        <Property name="PersistenceNamespace">
            <Value>com.topcoder.management.project.persistence.InformixProjectPersistence</Value>
        </Property>

        <Property name="ValidatorClass">
            <Value>com.topcoder.management.project.validation.DefaultProjectValidator</Value>
        </Property>

        <Property name="ValidatorNamespace">
            <Value>com.topcoder.management.project.validation.DefaultProjectValidator</Value>
        </Property>
    </Config>

<!-- Namespace for InformixProjectPersistence class -->
<Config name="InformixProjectPersistence.CustomNamespace">
    <!-- The DBConnectionFactory class name used to create DB Connection Factory, required -->
    <Property name="ConnectionFactoryNS">
        <Value>com.topcoder.db.connectionfactory.DBConnectionFactoryImpl</Value>
    </Property>
    <!-- the connection name to retrieve connection in DB Connection Factory, required -->
    <Property name="ConnectionName">
        <Value>informix_connection</Value>
    </Property>
    <Property name="ProjectIdSequenceName">
        <Value>project_id_seq</Value>
    </Property>
    <Property name="ProjectAuditIdSequenceName">
        <Value>project_audit_id_seq</Value>
    </Property>
</Config>

<!-- Namespace for UnmanagedTransactionInformixProjectPersistence class -->
<Config name="UnmanagedTransactionInformixProjectPersistence.CustomNamespace">
    <!-- The DBConnectionFactory class name used to create DB Connection Factory, required -->
    <Property name="ConnectionFactoryNS">
        <Value>com.topcoder.db.connectionfactory.DBConnectionFactoryImpl</Value>
    </Property>
    <!-- the connection name to retrieve connection in DB Connection Factory, required -->
    <Property name="ConnectionName">
        <Value>informix_connection</Value>
    </Property>
    <Property name="ProjectIdSequenceName">
        <Value>project_id_seq</Value>
    </Property>
    <Property name="ProjectAuditIdSequenceName">
        <Value>project_audit_id_seq</Value>
    </Property>
</Config>
<!-- Namespace for ApplicationsManagerImpl class -->
<Config name="com.topcoder.management.project.ApplicationsManagerImpl">
 <Property name="PersistenceClassName">
  <Value>com.topcoder.management.project.persistence.InformixReviewApplicationPersistence</Value>
 </Property>
 <Property name="PersistenceNamespace">
  <Value>com.topcoder.management.project.persistence.InformixReviewApplicationPersistence</Value>
 </Property>
</Config>
<!-- Namespace for InformixReviewApplicationPersistence class -->
<Config
       name="com.topcoder.management.project.persistence.InformixReviewApplicationPersistence">
 <Property name="DBConnFactoryNamespace">
  <Value>com.topcoder.db.connectionfactory.DBConnectionFactoryImpl</Value>
 </Property>
 <!-- the connection name to retrieve connection in DB Connection Factory, required -->
 <Property name="ConnectionName">
  <Value>dbconnection</Value>
 </Property>
 <Property name="ReviewApplicationIDSequenceName">
  <Value>review_application_id_seq</Value>
 </Property>
 <Property name="CacheNamespace">
  <Value>simpleCache</Value>
 </Property>
</Config>
<!-- Namespace for UnmanagedTransactionInformixReviewApplicationPersistence class -->
<Config
name="com.topcoder.management.project.persistence.UnmanagedTransactionInformixReviewApplicationPersi
stence">
 <Property name="DBConnFactoryNamespace">
  <Value>com.topcoder.db.connectionfactory.DBConnectionFactoryImpl</Value>
 </Property>
 <!-- the connection name to retrieve connection in DB Connection Factory, required -->
 <Property name="ConnectionName">
  <Value>dbconnection</Value>
 </Property>
 <Property name="ReviewApplicationIDSequenceName">
   <Value>review_application_id_seq</Value>
 </Property>
 <Property name="CacheNamespace">
   <Value>simpleCache</Value>
  </Property>
</Config>

<!-- Namespace for DBConnectionFactory component -->
<Config name="com.topcoder.db.connectionfactory.DBConnectionFactoryImpl">
    <Property name="connections">
        <Property name="default">
            <Value>informix_connection</Value>
        </Property>
        <Property name="informix_connection">
            <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-
://127.0.0.1:1526/project:INFORMIXSERVER=ol_informix</Value>
                </Property>
                <Property name="user">
                    <Value>informix</Value>
                </Property>
                <Property name="password">
                    <Value>11223344</Value>
                </Property>
            </Property>
        </Property>
    </Property>
</Config>

  <!-- Namespace for SearchBuilder component
    The setting in this section decides the logic of project searching.
   -->
  <Config name="com.topcoder.searchbuilder.ProjectManagement">
        <Property name="searchBundles">
            <Property name="ProjectSearchBundle">
                        <Property name="type">
                      <Value>Database</Value>
              </Property>
                        <Property name="name">
                      <Value>ProjectSearchBundle</Value>
              </Property>
                        <Property name="context">
                         <Value>SELECT DISTINCT
                   project.project_id, project_status_lu.project_status_id,
                   project_status_lu.name, project_category_lu.project_category_id,
                   project_category_lu.name, project_type_lu.project_type_id, project_type_lu.name,
                   project_type_lu.review_system_version, project.create_user, project.create_date,
                   project.modify_user, project.modify_date
                   FROM project
                   JOIN project_category_lu ON
                   project.project_category_id = project_category_lu.project_category_id
                   JOIN project_status_lu ON
                   project.project_status_id = project_status_lu.project_status_id
                   JOIN project_type_lu ON
                   project_category_lu.project_type_id = project_type_lu.project_type_id
                   LEFT JOIN project_info ON
                   project.project_id = project_info.project_id
                   LEFT JOIN project_info_type_lu ON
                   project_info.project_info_type_id = project_info_type_lu.project_info_type_id
                   LEFT JOIN resource ON
                   project.project_id=resource.project_id
                   LEFT JOIN resource_info ON
                   resource.resource_id = resource_info.resource_id
                   LEFT JOIN resource_info_type_lu ON
                   resource_info.resource_info_type_id =
                   resource_info_type_lu.resource_info_type_id WHERE</Value>
                </Property>

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

            <Property name="alias">
                <Property name="ProjectTypeID">
                    <Value>project_type_lu.project_type_id</Value>
                </Property>
                <Property name="ProjectTypeName">
                       <Value>project_type_lu.name</Value>
                </Property>
                <Property name="ProjectCategoryID">
                    <Value>project_category_lu.project_category_id</Value>
                </Property>
                <Property name="ProjectCategoryName">
                    <Value>project_category_lu.name</Value>
                </Property>
                <Property name="ProjectStatusID">
                    <Value>project_status_lu.project_type_id</Value>
                </Property>
                <Property name="ProjectStatusName">
                    <Value>project_status_lu.name</Value>
                </Property>
                <Property name="ProjectPropertyName">
                    <Value>project_info_type_lu.name</Value>
                </Property>
                <Property name="ProjectPropertyValue">
                    <Value>project_info.value</Value>
                </Property>
                <Property name="ProjectResourcePropertyName">
```

```
                    <Value>resource_info_type_lu.name</Value>
                </Property>
                <Property name="ProjectResourcePropertyValue">
                    <Value>resource_info.value</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

How to create a new instance of InformixProjectPersistence class:

```
// create the instance with the given namespace
ProjectPersistence persistence = new InformixProjectPersistence(
       "InformixProjectPersistence.CustomNamespace");
```

How to manage project using the ProjectPersistence instance:

```
// first create an instance of InformixProjectPersistence class
ProjectPersistence persistence = new InformixProjectPersistence(
       "InformixProjectPersistence.CustomNamespace");

// get all project categories from the persistence
ProjectCategory[] projectCategories = persistence.getAllProjectCategories();

// get all project statuses from the persistence
ProjectStatus[] projectStatuses = persistence.getAllProjectStatuses();

// project types can also be retrieved from the persistence, each
// project type can contains 0-n project category.
ProjectType[] projectTypes = persistence.getAllProjectTypes();

// get all project property types from the persistence.
ProjectPropertyType[] propertyTypes = persistence.getAllProjectPropertyTypes();

// create the project using a project category and a project status. The
// project id will be zero, which mean it isn't persisted
Project project = new Project(projectCategories[0], projectStatuses[0]);

// persist the project instance with the operator name "admin"
// the operator will be the creation/modification of this project
// instance
persistence.createProject(project, "admin");

// after creation, new values will be set to the project instance such
```

```
        // as project id, creation user, creation date, modification user,
        // modification date.

        // set new project category to the project instance
        project.setProjectCategory(projectCategories[1]);

        // update project with the operator "programmer"
        persistence.updateProject(project, "update reason", "programmer");

        // finally, we can get the project from the persistence.
        Project project1 = persistence.getProject(1);

        // we can also get a set of projects from the persistence in one method.
       Project[] projects = persistence.getProjects(new long[] { 1, 2, 3, 4 });
```

4.3.2 Manipulating project properties

```
    // to add a property to the project, first we have to get the defined property type
    // because only defined property name can be persisted
    ProjectPropertyType[] definedPropTypes = manager.getAllProjectPropertyTypes();

    // get a property names from defined types
    String definedName1 = definedPropTypes[0].getName();
    String definedName2 = definedPropTypes[1].getName();

    // add the properties to the project instance with some value
    project.setProperty(definedName1, "value1");
    project.setProperty(definedName2, "value2");

    // remove a property from the project by setting its value to null
    project.setProperty(definedName2, null);

    // update the project
    manager.updateProject(project, "programmer");
```

4.3.3 Search for projects using various conditions

```
    // Searching requires to build a search filter/
    // then pass it to the ProjectManagerImpl#searchProjects() method

    // build filter to search for project with category id = 1, similar methods are defined
    // for status id and type id
    Filter categoryIdFilter = ProjectFilterUtility.buildCategoryIdEqualFilter(1);

    // search for projects
    Project[] searchResult = manager.searchProjects(categoryIdFilter);

    // build filter to search for project with category name "Database", similar methods are defined
    // for status name and type name
    Filter categoryNameFilter = ProjectFilterUtility.buildCategoryNameEqualFilter("Database");

    // each type of filter has two format: EqualFilter and InFilter
    // to search for status id = 1,2,3, use this
    List statusIds = new ArrayList();
    statusIds.add(new Long(1));
    statusIds.add(new Long(2));
    statusIds.add(new Long(3));
    Filter statusIdsFilter = ProjectFilterUtility.buildStatusIdInFilter(statusIds);

    // search for project using its property name, value or combination
    // similar methods are defined for resource properties.
    // build filter to search for project which contains property name "testName"
    Filter propertyNameFilter =
ProjectFilterUtility.buildProjectPropertyNameEqualFilter("testName");

    // build filter to search for project which contains property with value "testValue"
    Filter propertyValueFilter =
ProjectFilterUtility.buildProjectPropertyValueEqualFilter("testValue");
```

```
    // build filter to search for project which contains property with name "testName" and value
"testValue"
    Filter propertyFilter = ProjectFilterUtility.buildProjectPropertyEqualFilter("testName",
"testValue");

    // filter can be combined to create new filters
    Filter projectType = ProjectFilterUtility.buildTypeIdEqualFilter(1);
    Filter projectStatus = ProjectFilterUtility.buildStatusIdEqualFilter(1);
    // create new filter using AND filter
    Filter andFilter = ProjectFilterUtility.buildAndFilter(projectType, projectStatus);
    // create new filter using OR filter
    Filter orFilter = ProjectFilterUtility.buildOrFilter(projectType, projectStatus);
    // create new filter using NOT filter
    Filter notFilter = ProjectFilterUtility.buildNotFilter(projectType);
```

4.3.4 Manage ReviewApplication entities

```
    // create an instance of ReviewerStatisticsManagerImpl
    ApplicationsManagerImpl manager = new
    ApplicationsManagerImpl(ApplicationsManagerImpl.class.getName());
    ReviewApplication ra = new ReviewApplication();

    // set properties of ReviewApplication
    ra.setApplicationDate(new Date(0));
    ra.setReviewerId(19);
    ra.setProjectId(1);
    ra.setAcceptPrimary(true);

    // create ReviewApplication
    ra = manager.create(ra);
    // ra has new ID 1

    // create another
    ReviewApplication rb = new ReviewApplication();
    rb.setApplicationDate(ra.getApplicationDate());
    rb.setReviewerId(20);
    rb.setAcceptPrimary(ra.isAcceptPrimary());
    rb.setProjectId(ra.getProjectId());
    rb = manager.create(rb);

    // rb has new ID 2
    // update ReviewApplication
    rb.setAcceptPrimary(false);
    rb = manager.update(rb);

    // retrieve ReviewApplication by id
    ReviewApplication rc = manager.retrieve(rb.getId());

    // rc should be the same as rb
    assertEquals(rb.getId(), rc.getId());

    // delete ReviewApplication
    assertTrue(manager.delete(rb.getId()));
    // rb is deleted

    // re-create rb
    rb = manager.create(rb);

    // rb has new ID 3
    // get primary applications
    ReviewApplication[] primaryApps = manager.getPrimaryApplications(1);
    // an array containing ra should be returned
    assertEquals(ra.getId(), primaryApps[0].getId());

    // get secondary applications
    ReviewApplication[] secondaryApps = manager.getSecondaryApplications(1);
    // an array containing rb should be returned
    assertEquals(rb.getId(), secondaryApps[0].getId());

    // get all applications
    ReviewApplication[] allApps = manager.getAllApplications(1);
```

```
// an array containing ra and rb should be returned.
// an array containing ra should be returned
assertEquals(ra.getId(), allApps[0].getId());
assertEquals(rb.getId(), allApps[1].getId());
```

4.3.4 Managed transaction demo

Version 1.1 of this component enables the use of externally managed transactions. One scenario a customer might use is an EJB application using container managed transactions.
A user wishing to take advantage of this new functionality in an EJB application will need to implement a Session Bean and a Home and Local or Remote interface. In this demo, a possible implementation of a Session Bean along with a Local Home and Local interface is shown, and a possible deployment descriptor that will ensure that all methods are executed in the scope of a transaction.

The following session bean uses the UnmanagedTransactionInformixProjectPersistence, allowing a client to create the project through a session bean with container managed transactions.
If a business operation fails, the setRollbackOnly() method is used to inform the EJB container that the current transaction should be rolled back at the appropriate time rather than committed.
ProjectBean.java:

```
package com.topcoder.management.project.persistence;

import javax.ejb.CreateException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

import com.topcoder.management.project.PersistenceException;
import com.topcoder.management.project.Project;

/**
 * The bean class used for EJB Demo.
 */
public class ProjectBean implements SessionBean {

    /**
     * Represents the session context.
     */
    private SessionContext context;

    /**
     * Represents the persistence class.
     */
    private AbstractInformixProjectPersistence persistence;

    /**
     * Creates the bean with the given namespace.
     * @throws CreateException if fails to create the bean.
     */
    public void ejbCreate() throws CreateException {
        try {
            persistence = new
UnmanagedTransactionInformixProjectPersistence("com.topcoder.management.p
roject.persistence");
        } catch (Exception e) {
            throw new CreateException("Fails to create the bean.");
        }
    }
    /**
     * sets the session context, called by EJB container.
     * @param ctx the session context to set.
     */
    public void setSessionContext(SessionContext ctx) {
        context = ctx;
    }

    /**
     * The business method to create the project.
     * @param project the project to create in database.
     * @param operator the operator who create the project.
     * @throws PersistenceException if fails to create the project in
database.
     * @throws IllegalArgumentException if any arguments is
<code>null</code>
     *             or the operator is empty string.
     */
    public void createProject(Project project, String operator)
        throws PersistenceException {

        if (project == null) {
            throw new IllegalArgumentException("The argument project
could not be null.");
        }
        if (operator == null) {
            throw new IllegalArgumentException("The argument operator
could not be null.");
        }
        if (operator.trim().length() == 0) {
            throw new IllegalArgumentException("The argument operator
could not be empty string.");
        }
        try {

            persistence.createProject(project, operator);

        } catch (PersistenceException e) {
            // if the operation fails, rollback the transaction
            context.setRollbackOnly();
            throw e;
        }
    }

    /**
     * It is called when the bean is removed.
     */
    public void ejbRemove() {
        System.out.println("ejbRemove");
    }

    /**
     * It is called when the bean is activated.
     */
    public void ejbActivate() {
        System.out.println("ejbActive");
    }
```

```
     /**
      * It is called when the bean is passivated.
      */
     public void ejbPassivate() {
         System.out.println("ejbPassivate");
     }
 }
```

In order to use the session bean implemented above, a client application will need an instance of its interface, which can be accessed through the home interface - the concrete implementations of these classes are generated by the EJB container.

ProjectHome.java:

```
package com.topcoder.management.project.persistence;


import java.rmi.RemoteException;


import javax.ejb.CreateException;

import javax.ejb.EJBHome;


/**

 * The EJB home used for EJB Demo.

 */

public interface ProjectHome extends EJBHome {


    /**

     * Create the <code>ProjectObject</code>.

     * @return the created <code>ProjectObject</code> object.

     * @throws RemoteException if there is anything remote problem.

     * @throws CreateException if fails to create the object.

     */

    ProjectObject create() throws RemoteException,

            CreateException;

}
```

ProjectObject.java:

```
package com.topcoder.management.project.persistence;


import java.rmi.RemoteException;


import javax.ejb.EJBObject;


import com.topcoder.management.project.PersistenceException;
```

```java
import com.topcoder.management.project.Project;

/**
 * The EJB Object used for the EJB demo.
 */
public interface ProjectObject extends EJBObject {

    /**
     * Creates the project with the given <code>Project</code> and operator.
     * @param project the project to create in database.
     * @param operator the operator who create the project.
     * @throws RemoteException if there is any remote problem.
     * @throws PersistenceException if fails to create the project in database.
     * @throws IllegalArgumentException if any arguments is <code>null</code>
     *            or the operator is empty string.
     */
    public void createProject(Project project, String operator)
        throws RemoteException, PersistenceException;

}
```

The deployment descriptor informs the EJB container of what enterprise beans are in use and how to manage the transactions.
Since the transaction attribute "required" is used for all business methods of ProjectBean, all method calls will occur within the scope of a transaction. If the client already has an open transaction, that transaction will be used; otherwise, a new transaction will be started.
Deployment descriptor:

```xml
<?xml version="1.0"?>

<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd"
  version="2.1">
  <enterprise-beans>
    <session>
        <ejb-name>Project</ejb-name>
        <home>com.topcoder.management.project.persistence.ProjectHome</home>
        <remote>com.topcoder.management.project.persistence.ProjectObject</remote>
        <ejb-class>com.topcoder.management.project.persistence.ProjectBean</ejb-class>
        <session-type>Stateless</session-type>
        <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
  <assembly-descriptor>
             <container-transaction>
                    <method>
                           <ejb-name>Project</ejb-name>
                           <method-name>*</method-name>
                    </method>
                    <trans-attribute>Required</trans-attribute>
             </container-transaction>
       </assembly-descriptor>
</ejb-jar>
```

By using container managed transactions, the behavior of the transaction management can easily be tweaked by editing the deployment descriptor, without altering any code. Enabling this pattern of usage is the only new functionality in version 1.1. Since the actual operations performed are identical to those shown in the existing demo, that part of the demo won't be repeated here.

The EJB Client

```
       try {

             // create the necessary properties.
             java.util.Properties p = new java.util.Properties();
             p.put(Context.INITIAL_CONTEXT_FACTORY,
                     "org.jnp.interfaces.NamingContextFactory");
             p.put(Context.URL_PKG_PREFIXES, "jboss.naming:org.jnp.interfaces");
             p.put(Context.PROVIDER_URL, "localhost:1099");

             // looks up the home
             Context jndiContext = new InitialContext(p);
             Object ref = jndiContext.lookup("ProjectHome");

             // create the home and object
             ProjectHome home = (ProjectHome) PortableRemoteObject.narrow(ref,
                     ProjectHome.class);
             ProjectObject pb = home.create();

             // create the Project by EJB
             pb.createProject(getProject(), "topcoder");
        } catch (Exception e) {
             e.printStackTrace();
        }
```

5. Future Enhancements

Additional persistence plug-in could be developed.
Additional validators could be supported to validate projects.