Project Management 1.0 Component Specification

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.

1.1 Design Patterns

Strategy pattern is used in ProjectValidator and ProjectPersistence interface. This allows plugging in different implementations of project validator and persistence.

Facade pattern is used in ProjectManagerImpl class to provide access to the component functions.
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 projects:

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 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*

Gets the project from the persistence before update using “**Get project using project_id**” routine. (oldProject)
Throw exception if the project with given id does not exist in the persistence.

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” 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,
project.project_category_id, project.create_user, project.create_date,
project.modify_user, project.modify_date, status.name, category.name
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
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"/LongValidator.isPositive()
    "ProjectCategoryID"/LongValidator.isPositive()
    "ProjectStatusID"/LongValidator.isPositive()
    "ProjectTypeName"/ StringValidator.hasLength(IntegerValidator.
lessThanOrEqualTo (64))
    "ProjectCategoryName"/ StringValidator.hasLength(IntegerValidator.
lessThanOrEqualTo (64))
    "ProjectStatusName"/ StringValidator.hasLength(IntegerValidator.
lessThanOrEqualTo (64)));
    "ProjectPropertyName"/ StringValidator.hasLength(IntegerValidator.
lessThanOrEqualTo (64)));
    "ProjectPropertyValue"/ StringValidator.hasLength(IntegerValidator.
lessThanOrEqualTo (4096)));
    "ProjectResourcePropertyName"/
StringValidator.hasLength(IntegerValidator. lessThanOrEqualTo (64)));
```

```
            "ProjectResourcePropertyValue"/
        StringValidator.hasLength(IntegerValidator. lessThanOrEqualTo (4096)));

        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);
            // get the number of result records.
            int length = result.getRecordCount();
            // this array is used to store ids.
            long[] ids = new long[length];
            // get the result
            for (int i = 0; i < length; i++) {
                ids[i] = result.getLong(0);
            }
            // use persistence to get projects.
            return persistence.getProjects(ids);
```

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));
```

1.4 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.

InformixProjectPersistence

This class contains operations to create/update/retrieve project instances from the Informix database. It also provides methods to retrieve database look up table values. It implements the ProjectPersistence interface to provide a plug-in persistence for Informix database. It is used by the ProjectManagerImpl class. The constructor takes a namespace parameter to initialize database connection.
Note that in this class, delete operation is not supported. To delete a project, its status is set to 'Deleted'. Create and update operations here work on the project and its related items as well. It means creating/updating a project would involve creating/updating its properties.

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

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 and InformixProjectPersistence.

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.

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.

2. Environment Requirements

2.1 Environment

JDK 1.3

2.2 TopCoder Software Components

- Configuration Manager v2.1.4 – used to read the configuration information.
- DB Connection Factory v1.0 – used to create the connection to the database.
- Search Builder v1.1 – 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.

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 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

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 WHERE
```

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</V
alue>
        </Property>

        <Property name="PersistenceNamespace">

<Value>com.topcoder.management.project.persistence.InformixProjectPersistence</V
alue>
        </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="com.topcoder.management.project.persistence.InformixProjectPersistence">
        <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</Value
>
                </Property>

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

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

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

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

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

  <!-- 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 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
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

```
    // create manager using default namespace
    ProjectManager manager = new ProjectManagerImpl();

    // project types, categories and statuses are pre-defined in the persistence
    // this component don't add/remove them

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

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

    // 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
    manager.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"
    manager.updateProject(project, "reason", "programmer");

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

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, "reason", "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);
```

5. Future Enhancements

Additional persistence plug-in could be developed.

Additional validators could be supported to validate projects.