Project Services 1.0 Component Specification

1. Design

The Project Services Component implements some of the business rules for combining Projects, Resources, Phases and Teams together. Specifically, it will provide streamlined access to project information. It will allow for a simple search for full or basic project information, or to use custom search criteria to locate projects, either in its full or basic form. The basic form involves getting the Project object (from Project Management), and the full form involves the FullProjectData object, which not only provides information as the basic form, but also provides project phase information, all resources participating in the project, and all teams currently existing in it. Furthermore, it provides data about the technologies involved in this project (such as Java, C#, etc…).

The service comes as an interface, which defines the business API for this component, and with an implementation.

The main interface of this component is fine grained enough to provide a useful API to client applications, but coarse grained enough to offer transactional atomic services and allow the presentation layer to minimize the calls to this layer.

This design makes use of the Logging Wrapper to facilitate informational and debugging logging in the services implementation. Logging is not performed inside bean implementations as there is no need to know such granular and trivial information. Logging is optional.

1.1 Design considerations

1.1.1 EJBs, ConfigManager, and Logging Wrapper

The EJB specification stipulates that File I/O is not allowed during the execution of the bean. At first glance this might mean that the use of the ConfigManager, and by extension Object Factory, could not be used because it performs property retrieval and storing using files. However, the restriction is only placed on the bean's lifetime, not the ConfigManager's. Therefore, as long as the ConfigManager does not perform I/O itself during the execution of the bean, or to be more accurately, that the thread performing a bean operation does not cause the ConfigManager to perform I/O, the use of ConfigManager to hold an in-memory library of properties is acceptable. Therefore, this design makes extensive use of ConfigManager and Object Factory, again, with the stipulation that the ConfigManager implementation does not perform I/O when this component is retrieving properties.

The issue of using Logging Wrapper is similar. The user of this component must ensure that logging is performed in a manner that does not violate EJB standards, such as not writing to a file or console.

1.2 Design Patterns

1.2.1 Strategy

This pattern is not really used in this design. Although ProjectServicesImpl uses the various external managers and services in a transparent manner, it does not introduce in this design anything new that is used in such a manner.

1.2.2 Façade

The **Façade** pattern is used in the ProjectServices, as it provides streamlined access to project data with full associated information.

1.3 Industry Standards

- EJB (specifically, to ensure this component is compatible with EJB restrictions, as defined in http://java.sun.com/blueprints/qanda/ejb_tier/restrictions.html).
- JavaBeans

1.4 Required Algorithms

1.4.1 Logging

This section is the central place to describe how logging is performed in lieu of stating this in each method in the ZUML.

All methods in ProjectServicesImpl have access to a Log and should log in the following manner:

- Method entrance and exit at INFO level.
- All exceptions and errors at ERROR level. This includes illegal arguments.
- All calls to external TopCoder classes (just to managers and services classes, not beans) at DEBUG level. This includes before the call and after the call.
- Any additional logging is at the developer's discretion.

Note that if the ProjectServicesImpl's log is null, no logging is to be performed.

1.5 Component Class Overview

1.5.1 com.topcoder.project.service

This package holds the main interface and value object of this component.

ProjectServices
This represents the interface that defines all business methods for project data retrieval services. It will provide streamlined access to project information. It will allow for a simple search for full or basic project information, or to use custom search criteria to locate projects, either in its full or basic form. The basic form involves getting the Project object (from Project Management), and the full form involves the FullProjectData object, which not only provides information as the

basic form, but also provides project phase information, all resources participating in the project, and all teams currently existing in it. Furthermore, it provides data about the technologies involved in this project (such as Java, C#, etc…).

It has one implementation: ProjectServicesImpl.

Thread safety: Implementations must be thread-safe from the point of view of their use. Implementations can assume that passed objects will be operated on by just one thread.

FullProjectData
This class is a DTO assembled to transfer all the data regarding a single project. It contains project header and phase information, as well as all the resources participating in the project, all teams currently existing in it, and all technologies involved in completing it.

This entity follows java bean conventions for defining setters and getters for these properties. Two constructors are also provided for convenient usage: One to match the super class Project, and another to set all fields in this class plus the two required fields for the super class.

Thread Safety: This class is mutable and not thread safe.

1.5.2 com.topcoder.project.service.impl

This package holds the implementation to the interface in the com.topcoder.project.service package.

ProjectServicesImpl
Full implementation of the ProjectServices interface. This implementation makes use of a large array of components to accomplish its task of retrieving project data: User Data Store Persistence component to retrieve external project information, the Project Phases and Phase and Resource Management to get project header and phase information, Resource Manager to get resource information, and Team Manager to get team information.

To provide a good view as the steps are progressing in each method, as well as full debug information for developers, the Logging Wrapper component is used in each method. To configure this component, the ConfigManager and ObjectFactory components are used.

Also provided are configuration-backed and programmatic constructors. This allows the user to either create all internal supporting objects from configuration, or to simply pass the instances directly.

Thread Safety: This class is immutable but operates on non thread safe objects, thus making it potentially non thread safe.

1.6 Component Exception Definitions

This component defines two custom exceptions. Note that all are runtime exceptions.

ProjectServicesException
Extends BaseRuntimeException. Represents a base exception for all other exceptions defined in this component. It is thrown by all ProjectServices methods.

ConfigurationException
Extends ProjectServicesException. Called by the ProjectServicesImpl constructors if a configuration-related error occurs, such as a namespace not being recognized by ConfigManager, or missing required values, or errors while constructing the managers and services.

1.7 Thread Safety

Thread safety is mentioned as a required part of this component. The component is virtually thread-safe, and under expected conditions, it is effectively thread-safe.

Objects such as FullProjectData are not thread-safe, and if had to be made thread-safe, its read/write operations would have to lock the fields they access.

Another aspect is the use of mutable, non-thread-safe objects in the ProjectServicesImpl methods. This effectively renders ProjectServicesImpl non-thread-safe and the only way to make it thread-safe is to ensure all objects it uses are thread-safe. However, it is expected that these bean instances will not be used by multiple threads, so this is not an issue.

In terms of transactional control, this component is assumed to work in an EJB environment. As such, it takes no steps to manually roll back any steps if errors are encountered.

2. Environment Requirements

2.1 Environment

- Development language: Java 1.4
- Compile target: Java 1.4

2.2 TopCoder Software Components

- Configuration Manager 2.1.5
    - Used for configuration of ProjectServicesImpl
- Object Factory 2.0.1

    - Used to obtain instances of required objects in ProjectServicesImpl. Some of the required objects are instances of PhaseManager, ResourceManager, etc…
- Base Exception 2.0
    - TopCoder standard for all custom exceptions.
- Logging Wrapper 2.0
    - Used for logging actions in ProjectServicesImpl. A Log instance is obtained from LogManager.
- Project Management 1.1
    - Provides the ProjectManager to retrieve Projects
- Resource Management 1.1
    - Provides ResourceManager to retrieve Resources. Also provides ResourceFilterBuilder to build resource filters.
- Project Phases 2.0
    - Provides the phase model: Project and Phase.
- User Project Data Store 2.0
    - Provides ProjectRetrieval to retrieve ExternalProjects.
- Phase Management 1.1
    - Provides PhaseManager to retrieve Projects from Project Phases 2.0.
- Team Management 1.0
    - Provides team information via the TeamManager interface
- Workdays 1.0
    - Provides the Workdays interface when a Project from Project Phases is mapped to FullProjectData.

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

There are no third party components that need to be used directly by this component.

3. Installation and Configuration

3.1 Package Names

com.topcoder.project.service
com.topcoder.project.service.impl

3.2 ConfigurationParameters

3.2.1 ProjectServicesImpl

Parameter	Description	Details
specNamespace	Namespace to use with the ConfigManagerSpecificationFactory. Required.	Example: "com.topcoder.specify"
userRetrievalKey	Key for the UserRetrieval to pass to ObjectFactory. Required	Valid key
resourceManagerKey	Key for the ResourceManager to pass to ObjectFactory. Required.	Valid key
phaseManagerKey	Key for the PhaseManager to pass to ObjectFactory. Required	Valid key
projectManagerKey	Key for the ProjectManager to pass to ObjectFactory. Required	Valid key
teamManagerKey	Key for the TeamManager to pass to ObjectFactory. Required	Valid key
loggerName	The name of the log to get from the LogManager. Optional	A valid name of the log. If not given, logging is not performed.
activeProjectStatusId	The ID of the active project status. Required	A non-negative long number.

3.3 Dependencies Configuration

3.3.1 TopCoder dependencies

The developer should refer to the component specification of the TopCoder components used in this component to configure them. Please see section 2.2 for a list of these components.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

None

4.3 Demo

4.3.1 Setup

Much of the setup for any demo involves properly configuring all the components. This demo will not go into that depth, but it will show a sample configuration XML file.

Here we define the configuration parameters in ConfigManager:

```
<Config name="TestConfig">
   <Property name="specNamespace">
         <Value>SpecificationNamespace</Value>
```

```
   </Property>
   <Property name="userRetrievalKey">
         <Value>userRetrievalKey</Value>
   </Property>
   <Property name="resourceManagerKey">
         <Value>resourceManagerKey</Value>
   </Property>
   <Property name="phaseManagerKey">
         <Value>phaseManagerKey</Value>
   </Property>
   <Property name="projectManagerKey">
         <Value>projectManagerKey</Value>
   </Property>
   <Property name="teamManagerKey">
         <Value>teamManagerKey</Value>
   </Property>
   <Property name="loggerName">
         <Value>defaultLogger</Value>
   </Property>
   <Property name="activeProjectStatusId">
         <Value>1</Value>
   </Property>
</Config>
```

For the demo, we will assume that an active project status has an ID of 1.

For the purpose of the demonstration, we assume the following abridged data is available to the service. Each item number represents the ID, not a quantity.

Project 1: Active, Type=Design

- Resources: 1,2
- Teams: 1
- Technologies: C#, SQL

Project 2: Active, Type=Development

- Resources: 3,4,5,6
- Teams: 2,3
- Technologies: C#, XML

Project 3: InActive, Type=Design

- Resources: 1,5
- Teams: 4
- Technologies: Java, EJB

The construction of the services class would be done in the following manner.

```
// Create ProjectServicesImpl from configuration.
ProjectServices service = new ProjectServicesImpl(“TestConfig”);
```

4.3.2 *Usage*

A typical usage scenario involves querying for project information: We use the service instance and scenario setup from 4.3.2.

```
// Find active project headers
Project[] activeProjectHeaders = service.findActiveProjectsHeaders();
// This would return 2 projects: 1, 2

// Find active projects
FullProjectData[] activeProjects = service.findActiveProjects();
// This would return 2 projects: 1, 2.
// Project 1 would have resources 1 and 2, teams 1, and technologies C#
and SQL.
// Project 2 would have resources 3, 4, 5, and 6, teams 2 and 3, and
technologies C# and XML.

// Find project headers. This would involve using Filters. We will
perform a search for all Design projects.
Filter typeFilter = // filter to retrieve projects with type=Design
Project[] projectHeaders = service.findProjectsHeaders(typeFilter);

// This would return 2 projects: 1, 3
// The findFullProjects method would work in the same manner, but
retrieve the full project data

// Finally, one can retrieve data for a single project, by project ID
FullProjectData project = service.getFullProjectData(2);
// This would return Project 2, and would have resources 3, 4, 5, and 6,
teams 2 and 3, and technologies C# and XML.

// This demo has provided a typical scenario for the usage of the
service.
```

5. Future Enhancements

New services will be added to support more generic filters