Project Phase Template 1.0 Component Specification

1. Design

A project is usually executed in a predefined set of phases for a particular customer. Requiring the user to manually define the phase hierarchy is laborious and unnecessary. The component provides a template mechanism to handle this scenario. Template storage is pluggable and can be added without code changes. An XML storage is provided with this release.

There're two things need to be pluggable in this component - the template persistence logic and the default project start date generation logic. Two interfaces are defined to fulfill those requirements, so that plugging a persistence mechanism or start date generation algorithm is very simple.

PhaseTemplatePersistence interface defines the contract to generate a set of project phases from a specific persistence mechanism, a built-in implementation based on XML persistence is designed as the default persistence logic, a schema and a sample persistence file are provided in the docs directory. Inside the XmlPhasePersistence, the template is stored as a Map of Document DOM objects, keyed on the template names.

StartDateGenerator interface defines the contract to generate a default project start date according to a specific generation algorithm, a built-in implementation which generates a relative time in a week is designed as the default start date generation logic, it can generate a relative time in any week based on the configurations. One thing to note is that, a concept of "week offset" is introduced to describe the relationship of weeks. For example, if we say the target week has a week offset of 1 from current week, then the target week is the next week, if we say the target week has a week offset of 0 from current week, then the target week is current week itself.

1.1 Design Patterns

Strategy Pattern - The component has decoupled the logic for phase template persistence(PhaseTemplatePersistence interface and its implementations) and project start date generation(StartDateGenerator interface and its implementations).

1.2 Industry Standards

XML, XSD schema

1.3 Required Algorithms

1.3.1 Generate project phases from XML-based phase template persistence

The XML schema for the template persistence is fairly simple, there're three main concepts:

- **PhaseType definitions** : defines the phase types information needed in this template, type id and type name is included(note that in this XML schema, attribute "id" is reserved for internal reference, so attribute "typeId" is used to store the id of the type, while "id" attribute is used to store the internal reference point so that the phase definitions can specify the type conveniently).

- **Phase definitions** : defines the phases in this template, the information included are – phase length(in “length” attribute), phase type(in “type” attribute”) and dependency information(in “Dependency” child elements).An internal “id” is reserved for internal reference so that we can specify the dependency information conveniently.
- **Dependency descriptions** : defines a single dependency of a phase, the information included are – the dependency phase(in “id” attribute), whether the dependency is a start(in “isDependencyStart” attribute, optional, default to false if missing) , whether the dependent is a start(in “isDependentStart” attribute, optional, default to true if missing) and lag time(in “lagTime” attribute, optional, default to 0 if missing) between the dependency and dependent. Note that zero or more “Dependency” child elements may exist, each one describes a single dependency information.

Please see docs/xml_phase_template.xsd and docs/sample.xml for more details.

Inside the XmlPhasePersistence, the template is stored as a Map of Document DOM objects, keyed on the template names.

Given a template name, the phases generation algorithm is as following:

XmlPhaseTemplatePersistence.generatePhases(String templateName , Project project)

0. Getting the template DOM Document.

First, retrieve the DOM Document object for the template associated with the given templateName.

Document templateDOM = (Document) templates.get(templateName);

1. Phase types generation

Before generating the phases, all phase types defined in the template will be generated and cached in a Map with the id for further reference.(NOT typeId) as the key(The developer may choose lazy generation to improve the efficiency):

For each "PhaseType" element, create a PhaseType object with the "typeId" and "typeName" attributes, then put the PhaseType object into a map for caching.

```
Map phaseTypes = new HashMap();
NodeList list = templateDOM.getElementsByTagName("PhaseType");
for (int i = 0; i < list.getLength(); i++) {
   Element typeElement = (Element) list.item(i);
   phaseTypes.put(typeElement.getAttribute("id"),
    new PhaseType(Long.parseLong(typeElement.getAttribute("typeId")),
   typeElement.getAttribute("typeName"));
}
```

2. Phases generation

It is recommended to perform a 2-pass scan to generate the full hierarchy(however the developer may choose more efficient approach where appropriate):

In the 1st pass, create a Phase object for each "Phase" element with the "length" and "type" attributes, ignore the dependencies information, and cache the Phase objects in a Map, with the "id" attribute as the key.

In the 2nd pass, go through the dependencies information for each "Phase" element, add the dependency relationship to the created Phase objects.

Finally, return the created phase objects.

```
// 1st pass
Map phases = new HashMap();
list = templateDOM.getElementsByTagName("Phase");
for (int i = 0; i < list.getLength(); i++) {
    Element phaseElement = (Element) list.item(i);
   long length = Long.parseLong(phaseElement.getAttribute("length"));
    Phase phase = new Phase(project, length);
    phase.setPhaseType((PhaseType)
            phaseTypes.get(phaseElement.getAttribute("type")));
    phases.put(phaseElement.getAttribute("id"), phase);
}
// 2nd pass
for (int i = 0; i < list.getLength(); i++) {
    Element phaseElement = (Element) list.item(i);
    NodeList dependencyElements =
             phaseElement.getElementsByTagName("Dependency");
    Phase phase = (Phase) phases.get(phaseElement.getAttribute(“id”));
    for (int j = 0; j < dependencyElements.getLength(); j++) {
        Element dependencyElement = (Element) dependencyElements.item(j);
        Phase dependencyPhase = (Phase)
                phase.get(dependencyElement.getAttribute(“id”));
        // isDependencyStart flag, optional attribute, default to false if missing.
        boolean isDependencyStart = false
        // isDependentStart flag, optional attribute, default to true if missing
        boolean isDependentStart = true;
        // lagTime between the dependent and the dependency, optional attribute, default to
        //0  if missing
        int lagTime = 0;
        // temp variable to cache attribute value
        String tmp = dependencyElement.getAttribute("isDependencyStart");
```

```
            if (tmp != null) {
                if (tmp.equals(“true”) {
                    isDependencyStart = true;
                 } else if (tmp.equals(“false”) {
                    isDependencyStart = false;
                 } else {
                    throw new IllegalArgumentException(“can not parse isDependencyStart”);
                 }

            }
            tmp = dependencyElement.getAttribute("isDependentStart");
            if (tmp != null) {
                 if (tmp.equals(“true”) {
                    isDependentStart = true;
                 } else if (tmp.equals(“false”) {
                    isDependentStart = false;
                 } else {
                    throw new IllegalArgumentException(“can not parse isDependentStart”);
                 }
            }
            tmp = dependencyElement.getAttribute("lagTime");
            if (tmp != null) {
                 lagTime = Integer.parseInt(tmp);
            }
            Dependency dependency = new Dependency(dependencyPhase,
                               phase,  isDependencyStart, isDependentStart, lagTime);
            phase.addDependency(dependency);
        }
    }
    // add the phases to the project
    for (Iterator itr = phases.values().iterator(); itr.hasNext();;) {
        project.addPhase((Phase) itr.next());
    }
```

If any exception is caught in the above process, wrap it into PhaseGenerationException and throw out.Generally the possible errors would be :(1) dependency reference doesn't exist(2) phase type reference doesn't exist (3)cyclic dependency exists in the template(certain exception will be thrown from Project Phases component).

1.3.2 Generate a relative time in a week

1. Obtain a Calendar instance for current time:

Calendar cal = Calendar.getInstance();

2. Adjust the time to the week according to the weekOffset

cal.add(Calendar.WEEK_OF_YEAR, weekOffset);

3. Adjust the time to the day of week according to the dayOfWeek

cal.set(Calendar.DAY_OF_WEEK, dayOfWeek);

4. Adjust the time to the exact time according to the hour, minute, second.

cal.set(Calendar.HOUR_OF_DAY, hour);

cal.set(Calendar.MINUTE, minute);

cal.set(Calendar.SECOND, second);

5. Return the adjusted time.

return cal.getTime();

1.3.3 Generate a project with phases from a given template and a start date

DefaultPhaseTemplate.applyTemplate(String template, Date startDate):

1. Create an empty project : *Project project = new Project(startDate, workdays);*
2. Generate a set of phases and add them to the project by calling *persistence.generatePhases(template, project);*

1.3.4 Generate a project with phases from a given template, without a specific start date provided

DefaultPhaseTemplate.applyTemplate(String templat):

1. Generate a start date by calling *startDateGenerator.generateStartDate(),*
2. Create an empty project with the date generated from 1 :

   Project project = new Project(startDate, workdays);
3. Generate a set of phases and add them to the project by calling *persistence.generatePhases(template, project);*

1.4 Component Class Overview

Note: a more thorough description of all the classes is available in the Documentation tabs within Poseidon.

Interface PhaseTemplate
PhaseTemplate interface defines the contract to access the phase templates, generally the implementations will manage several different templates which are used to generate different project phase hierarchies. It provides the API to generate a set of project phases from a given predefined template, and compose a Project object with those phases, with or without a specified project start date, it also provides the API to retrieve

names of all templates available to use.

Class DefaultPhaseTemplate
DefaultPhaseTemplate is a default implementation of PhaseTemplate interface.
It manages two underlying variables - persistence(of type PhaseTemplatePersistence) which is used as the actual template storage logic; and startDateGenerator(of type StartDateGenerator) which is used as the default project start date generation logic if there's no specific start date provided during the phase generation. By this, we can easily change the persistence and start date generator implementation so that the client is able to swap out for different storage and start date generation strategies without code changes - all we need to do is adding new PhaseTemplatePersistence and/or StartDateGenerator implementations.

Interface PhaseTemplatePersistence
PhaseTemplatePersistence interface acts as the persistence layer of the phase templates, so that the persistence is pluggable and can be added without code changes. It manages a set of templates, provides the API to generate an array of Phases from a template it manages.
Note that this interface generates only phases, the Project generation is out of the scope of this interface.
In this initial release, the persistence is read-only, all templates are not modifiable with this component.The template authoring functionalities may be added in the future versions.

Class XmlPhaseTemplatePersistence
XmlPhaseTemplatePersistence is an XML based persistence implementation.
Phase templates are stored in XML files, each XML file defines one template, the XML schema is defined in docs/xml_phase_template.xsd, and also a sample template is provided in docs directory.
Each template is assigned a template name, inside the XmlPhaseTemplatePersistence, templates are stored in a Map with the template name as the key, and with org.w3c.dom.Document objects parsed from the XML document as the value.

Interface StartDateGenerator
StartDateGenerator interface defines the contract to generate a default start date for a project according to specific generation logic, so that the DefaultPhaseTemplate can employ different start date generation logic without code changes.
It will be used in the DefaultPhaseTemplate as the default project start date generation logic if no start date specified in the phase generation process.

Class RelativeWeekTimeStartDateGenerator
RelativeWeekTimeStartDateGenerator is the initial built-in implementation of StartDateGenerator interface.
It generates a relative time in a week as the default start date, for example, it can be configured to generate 9:00 AM next Thursday as the default start date.

1.5 Component Exception Definitions

IllegalArgumentException
It may be thrown from many places in this component where the argument is null or empty string, or the numeric argument is out of range.

PhaseTemplateException:
It is the base class for custom exceptions thrown from this component.

PhaseGenerationException:
It may be thrown from PhaseTemplatePersistence and PhaseTemplate implementations if there're errors in the phase generation process, e.g. the cyclic dependency, reference to an undefined project phase, etc.

StartDateGenerationException:
It may be thrown from StartDateGenerator implementations if there're errors in the start date generation process, e.g. connection errors in database-based start date generation implementations. In the initial release, this exception is not used at all, it is designed for future extension.

ConfigurationException:
It may be thrown from many places where the outside configuration is needed but there're errors in the configuration. for example, the expected namespace is not loaded, the required configuration property is missing, etc.

PersistenceException
PersistenceException indicates that there're errors while accessing the template persistence, e.g. persistence file doesn't exist for file-based persistence layer.

1.6 Thread Safety

This component should be thread safe, implementations of the interfaces defined in this component must be thread safe.

RelativeWeekTimeStartDateGenerator and XmlPhaseTemplatePersistence are immutable and therefore they're thread safe.

DefaultPhaseTemplate is mutable(persistence and startDateGenerator may be modified), so appropriate locking mechanism is required in the applyTemplate method implementations, however it is fairly simple : lock on the persistence while generating phases from a template, lock on the startDateGenerator while generating the default project start date, lock on this in while changing the object references in setters.

2. Environment Requirements

2.1 Environment

Development language: Java 1.4
Compile target: Java 1.4, Java 1.5

2.2 TopCoder Software Components

- **Configuration Manager Version 2.1.4**
- **Project Phases Version 2.0**
- **Object Factory Version 2.0**
- **Workdays Version 1.0**

- **Base Exception Version 1.0**

2.3 Third Party Components

- **JAXP 3.0 Implementation(Xerces2-J , version 2.8.0 for example)**

3. Installation and Configuration

3.1 Package Name

com.topcoder.project.phases.template
com.topcoder.project.phases.template.persistence
com.topcoder.project.phases.template.startdategenerator

3.2 Configuration Parameters

3.2.1 Configuration for XmlPhaseTemplatePersistence

Parameter	**Description**	**Values**
template_files	A list of template files from which the templates will be loaded	XML persistence file names. **Required**

3.2.2 Configuration for RelativeWeekTimeStartDateGenerator

Parameter	**Description**	**Values**
week_offset	The week offset of the date to generate, from current week.	Any integer value.e.g. 0 means current week, 1 means the next week **Required**
day_of_week	the day of the week	Integer values from 1 to 7, representing SUNDAY to SATURDAY respectively. **Required.**
hour	hour in 24-hour clock	Integer values in [0, 23]. **Required**
minute	Minute value	Integer values in [0, 59] **Required**
second	Second value	Integer values in [0, 59] **Required**

3.2.3 Configuration for DefaultPhaseTemplate

Parameter	Description	**Values**
persistence	Configuration property for persistence, should have “class” and “namespace” sub-properties	See persistence.class and persistence.namespace
persistence.class	class of the PhaseTemplatePersistence	Full qualified class name **Required**
persistence.namespace	namespace from which the PhaseTemplatePersistence will be created	The configuration namespace **Optional**, if not provided, no arg ctor will be used to create the persistence
start_date_generator	Configuration property for start date generator, should have “class” and “namespace”	See start_date_generator.class and start_date_generator.namespace

	sub-properties	
start_date_generator.class	class of the StartDateGenerator	Full qualified class name **Required**
start_date_generator.namespace	namespace from which the StartDateGenerator will be created	The configuration namespace **Optional**, if not provided, no arg ctor will be used to create the startDateGenerator.
workdays	Configuration property for workdays variable, should have "object_specification_namespace" and "object_key" sub-properties, "object_identifier" sub-property is optional.	See workdays. object_specification_namespace, workdays.object_key and workdays.object_identifier
workdays. object_specification_namespace	Configuration namespace of the object specification configuration of the workdays	The configuration namespace. **Required**
workdays.object_key	key of the workdays object	The key of the workdays object in the Object specification configuration, **Required**
workdays.object_identifier	Identifier of the workdays object	The key of the workdays object in the Object specification configuration, **Optional.** If missing, the workdays will be created without identifier.

3.2.4 Object Specification Configuration for Workdays

```
<CMConfig>
 <Config name="com.topcoder.project.phases.template.DefaultPhaseTemplate.workdays">
  <Property name="workdays:default">
   <Property name="type">
    <Value>com.topcoder.date.workdays.DefaultWorkdays</Value>
   </Property>
   <Property name="params">
    <Property name="param1">
     <Property name="type">
      <Value>String</Value>
     </Property>
     <Property name="value">
      <Value>test_files/workdays.xml</Value>
     </Property>
    </Property>
    <Property name="param2">
     <Property name="type">
      <Value>String</Value>
     </Property>
     <Property name="value">
      <Value>XML</Value>
     </Property>
    </Property>
   </Property>
  </Property>
 </Config>
</CMConfig>
```

3.3 Dependencies Configuration

None.

4. Usage Notes

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

4.1 Required steps to test the component

Preload the configuration file into Configuration Manager. Follow demo.

4.2 Required steps to use the component

Preload the configuration file into Configuration Manager. Follow demo.
Please see the configuration files in docs directory for information on how to configure this component.

4.3 Demo

4.3.1 Create the PhaseTemplate instance from configuration

```
DefaultPhaseTemplate template = new
  DefaultPhaseTemplate("com.topcoder.project.phases.template.DefaultPhaseTemplate
  ");
```

4.3.2 Create the PhaseTemplate instance with specific PhaseTemplatePersistence and StartDateGenerator.

```
// create PhaseTemplatePersistence instance
PhaseTemplatePersistence persistence = new
  XmlPhaseTemplatePersistence("com.topcoder.project.phases.template.persistence.X
  mlPhaseTemplatePersistence");
// create StartDateGenerator instance
StartDateGenerator startDateGenerator = new
  RelativeWeekTimeStartDateGenerator("com.topcoder.project.phases.template.startda
  tegenerator.RelativeWeekTimeStartDateGenerator");
 // create PhaseTemplate instance with the persistence and startDateGenerator
 DefaultPhaseTemplate template = new DefaultPhaseTemplate(persistence,
   startDateGenerator, new DefaultWorkdays());
```

4.3.3 Apply a template to generate project phases

```
// apply a template with name "TCS Component Project" with a given start date
Project project = template.applyTemplate("TCS Component Project",
  Calendar.getInstance().getTime());
// apply a template with name "TCS Component Project" without a specific start date
project = template.applyTemplate("TCS Component Project");
```

4.3.4 Retrieve names of all available templates

```
// retrieve names of all available templates
String[] templateNames = template.getAllTemplateNames();
```

4.3.5 Change template persistence or start date generator dynamically

```
// change template persistence(say we have a SQLPhaseTemplatePersistence
template.setPersistence(new
  SQLPhaseTemplatePersistence("com.topcoder.project.phases.template.persistence.SQ
  LPhaseTemplatePersistence"));
// change start date generator(say we have a RelativeMonthTimeStartDateGenerator
```

template.setStartDateGenerator(new RelativeMonthTimeStartDateGenerator("com.topcoder.project.phases.template.startdategenerator.RelativeMonthTimeStartDateGenerator"));

5. Future Enhancements

- Implement new PhaseTemplatePersistence implementations to add new persistence(storage) mechanism, e.g. RDBMS-based persistence.
- Implement new StartDateGenerator implementations to add new project start date generation logic.
- Add template authoring/editing functionality
- Enhance the XmlPhaseTemplatePersistence to support dynamic template management(adding/removing).