# Requirements Specification

## 1. Scope

### 1.1 Overview

The existing Job Scheduling Component will be broken into two new components. Each Component working together to produce the functionality the original component performed, plus additional scheduling enhancements. The first of the two new components will enable the user the ability to define, edit and read a schedule. This schedule can be used by the second new Job Scheduling component as well as other components in the future that require scheduling.

### 1.2 Existing Requirements

*1.2.1 Scheduled Processing*

- Execute processes at a configured date, time and recurrence.
- Tasks can be scheduled at the following intervals: seconds, minutes, hours, days, weeks, months, years.
- Multiple entries for each time element must also be supported.

*1.2.2 Task Execution*

- Execute defined classes. (If necessary, the component can define a required interface for any classes that are to be executed to implement)
- Execute OS-level executables/scripts.

1.2.3 Task dependency

Allow a task to be dependent on another task regarding execution time. For example Task B does not have a configured datetime. It is configured to run on the successful completion of Task A. Allow the option to execute Task B even if Task A fails. Also allow configuration of an option time delay before task B executes after Task A.

A task setup must have either a scheduled datetime or a task dependency not both.

A circular task dependency should not be allowed.

1.2.4 Alert Notification

Allow set up of an email alert if a task returns a failure or cannot be executed. Configuration is at the task or task group level.

1.2.5 Task Group

A task group is a grouping of tasks to simplify job scheduler configuration. A group can contains one or more tasks.

### 1.3 Enhancements

1.3.1 Two components, There will be one component to manage the scheduling and a separate component to manage the execution of the schedule. The second component

will make use of the first to interact with the schedule. The designs will be independent of each other and will be developed in separate competitions.

1.3.2 Additional Scheduling behavior. The scheduling component will support the existing scheduling options (a cron entry) as well as the following new options.

- Each entry which is not a cron entry will have the following two fields
    - Start Date – a task will not run before this date. This field is optional.
    - Start Time – the time on the Start date to start. If the Start date is left open then it will only apply to times in the day after the start time on any day.
- An Entry will be defined to execute on,
    - Every Day – Provided a start time is provided the task will execute every day at that time on and after the start date (if provided)
    - Weekly
        - ♣ Week Day – this would mean that the task will run Monday through Friday
        - ♣ Week End – this would mean that the task will run on Saturday and Sunday
        - ♣ Specific Days of the Week – The user can specify any combination of days with in the week
    - Monthly
        - ♣ On a specific day of the month
        - ♣ On a specific {First, second, … last} {Day, weekday, Sunday, Monday…Saturday} of every X Month , for example it would run "The [Second] [Tuesday] of every [2] Months"
    - Yearly
        - ♣ On a specific Day of the year
        - ♣ On a specific {First, second, … last} {Day, weekday, Sunday, Monday…Saturday} of {Jan, Feb …. Dec}, for example it would run "The [First][Monday] of [September]
- An entry can recur for X number of times
- An Entry can be enabled or disabled, disabled tasks will not be executed

1.3.3 Multiple sources must exist for the persistence of the configuration
- Text File – this will be to support existing applications which only care about read only and/or don't have or desire access to a database.
- Database – allows an enhanced method of data storage. This is primarily the mode of operation for read/write applications.

1.3.4 The Scheduling component will have a read and write API which allows users to add entries through a UI. The UI is not part of this component but the API which allows it is.
- Add Scheduled Task – This will allow the user to create a new scheduled task. The entry will be persisted.
- View Scheduled Task – this will allow a user to view a specific task.
- Edit Scheduled Task – Allows a user to edit a specific task. The changes will be persisted.
- Enumerate Scheduled Task – returns the entire list of scheduled tasks.
- Remove Scheduled Task – removes a scheduled task. The Task will be removed from the persistence.

### 1.4 Example of the Software Usage

Scheduled Maintenance:

A typical use of cron is to execute a weekly script that deletes old files from a server during "off" hours. Implementing this functionality through the Cron component and Java classes will greatly increase the portability of the base level script, as well as the application that it is maintaining. This Component will assist cron like applications to manage schedules.

The new scheduling functionality is similar to that found in MS Outlook and other such tools. This allows a broader range of scheduling options which occur on a larger time scale.

## 2. Interface Requirements

*2.1.1 Graphical User Interface Requirements*

None.

*2.1.2 External Interfaces*

None specified.

*2.1.3 Environment Requirements*

- Development language: Java1.4
- Compile target: Java1.4, Java1.5

*2.1.4 Package Structure*

com.topcoder.util.scheduler.scheduling – this is the package for the scheduling component
com.topcoder.util.scheduler.processor – this is the package for the component that executes the tasks.

## 3. Software Requirements

### 3.1 Administration Requirements

*3.1.1 What elements of the application need to be configurable?*

Execution details for scheduled processing.

### 3.2 Technical Constraints

*3.2.1 Are there particular frameworks or standards that are required?*

None.

*3.2.2 TopCoder Software Component Dependencies:*

Configuration Manager 1.2
Executable Wrapper 1.0 (where possible)
Alert Factory

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

*3.2.3 Third Party Component, Library, or Product Dependencies:*

None.

*3.2.4 QA Environment:*

- Solaris 7
- RedHat Linux 7.1
- Windows 2000

**3.3 Design Constraints**

*3.3.1 Development Standards:*

- It is required that all code be clearly commented.
- All code must adhere to javadoc standards, and, at minimum, include the @author, @param, @return, @throws and @version tags.
    - When populating the author tag, use your TopCoder member handle. In addition, please do not put in your email addresses.
    - Copyright tag: Copyright © 2002-2006, TopCoder, Inc. All rights reserved
- For standardization purposes, code must use a 4-space (not tab) indentation.
- Do not use the ConfigurationManager inside of EJB's. The usage of the java.io.* package to read/write configuration files can potentially conflict with a restrictive security scheme inside the EJB container.
- All code must adhere to the Code Conventions outlined in the following: http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

*3.3.2 Database Standards:*

- Table and column names should be as follows: table_name, not tableName.
- Every table must have a Primary Key.
- Always use Long (or database equivalent) for the Primary Key.

For maximum portability, database objects (table names, indexes, etc) should be kept to 18 characters or less. Informix and DB2 have an 18-character limit.

**3.4 Required Documentation**

*3.4.1 Design Documentation*

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification
- Test Cases

*3.4.2 Help / User Documentation*

- Javadocs must provide sufficient information regarding component design and usage.