[TopCoder]

Job Scheduling 3.1 Requirements Specification

1. Scope

1.1 Overview

The 3.0 version has several flaws that limit its usage and functionalities, as this update will fix the issues and improve its usage. It also adds support to the new Configuration API component.

Note: Backward compatibility is required for this update.

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.

1.2.6 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.

1.2.7 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

1.2.8 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.

1.2.9 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.

1.3 Enhancements

1.3.1 Currently in the 3.0 version, there is no way to let job instances of a same class track states when being scheduled, if information (like the last scheduled date) needs to be exchanged between job instances created in a sequence, the current scheduling mechanism does not support it. Each time the Scheduler will create new job instance in such a manner that no additional information can be specified, this update will solve this issue.
    - Add API to the Job class to make it be able to receive information after initialization.
    - Provide a way for the ScheduledEnable interface to help it get states information from the Job instance which contains it.
    - The Scheduler implementations will be updated to be able to transfer the states information between the Job instances when scheduling.

1.3.2 When scheduling jobs that are java classes the Scheduler now can only create new instances by reflection using the default constructor, which limits the flexibility, this will be refactored to provide more options when creating jobs:

- Use Object Factory to support multi-argument constructor.
- Use *Factory Pattern* to create job instances. The factory interface will be defined by the designer and the component will be able to configure the factory instance(s) programmatically.
- Both options will be supported without any conflict, exceptions will be handled, and any errors will be logged if possible.

1.3.3 Currently there is no way to access the runningJob/runningObject from Job class, when the job type is Java class this is hard to use, e.g. the user may want to retrieve the job from Scheduler and do some configuration work to its runningJob/runningObject such as registering listeners or setting properties, this update will provide this feature.

- Provide an API for the Job class to access the runningJob/runningObject field. This capability must not break the original component's thread safety.

1.3.4 A new implementation of Scheduler that uses the new Configuration API component will be provided.

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.

A possible scenario of this update is, when scheduling a job that checks RSS feeds for update periodically, the Scheduler will create a new Job instance each time do the check. Then each Job instance needs to track the last check date to determine whether a RSS feed has newly published items, and after retrieving the newly updated feeds the user may need to be notified. A possible way to achieve this is to use observer pattern, then the user needs to access the Job's runningJob/runningObject from the Scheduler to register the listeners if he wants to do it programmatically.

1.5 Future Component Direction

None at this moment.

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, 1.5

2.1.4 Package Structure

com.topcoder.util.scheduler.scheduling

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 2.1.5
Configuration API 1.0
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
- Windows 2003

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

3.4 Required Documentation

3.4.1 Design Documentation

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

3.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the 'Documentation' tab of Poseidon.