Software Documentation : Java Workdays 1.1

This page last changed on Dec 14, 2010 by volodymyrk.

1. Scope

1.1 Overview

The Workdays Component provides a set of generic functions that perform various time calculations on a pre-defined workday schedule. A workday schedule is defined thorough a configuration file allowing for easy changes.

The version 1.1 adds support for the negative time amount to be passed to the add() method. When negative time amount is passed to the add() method it should correctly compute the resulting date by subtracting the time difference instead of adding it. It is important that the signature of the methods is not changed, the component must be backwards compatible.

Also, the support for the Configuration API component should be added. Initializing with Configuration Manager should still be supported for compatibility but should be marked as deprecated.

The designers are also responsible for updating the documentation and diagrams to the current standards. It is also in scope of the design to incorporate any code changes that could have been done to the code after the last design.

1.2 Logic Requirements

1.2.1 Setting Non-Workdays

- Ability to set holidays and other non-work days
- Ability to set whether or not weekend days are to be included as a normal workday

1.2.2 Setting Workday Times

- Ability to set the start and end hours of a work day (for example, work day starts at 8:00AM and ends at 5:30PM)

1.2.3 Date/Time Functions

- Function to add days, hours or minutes to an existing date and return the date that specifies how many work days it would take to complete. Examples:
  - Only taking weekends as non-work days into affect; 9:00AM - 5:00PM is normal work hours. Start Time: Friday December 3rd, 2004 10:00AM, add 33 hours and you would get an end time of: Thursday December 9th, 2004 11:00AM
  - Saturdays are a normal work day and workdays are from 8:00AM - 6PM. Start Time: Friday December 3rd, 2004 10:00AM, add 33 hours and you would get an end time of: Monday December 7th, 2004 1:00PM.

In version 1.1 negative amount of time units can be specified which means it is to be subtracted from the date.

1.2.4 Loading initial setup from configuration file

- Ability to load the initial non-workdays, weekend preferences, workday start time, and workday end time from a configuration file using the ConfigurationManager component.

In version 1.1 support for the Configuration API is added. Configuration Manager is still supported but marked as deprecated.

1.3 Required Algorithms

None required.

1.4 Example of the Software Usage

A project management application could use this component to figure the end date of a particular task when given the hours estimate from a developer.

1.6 Future Component Direction

Any enhancement needs to be approved either in forum or in email with managers to eliminate over-complicating the component with useless functions.

2. Interface Requirements

2.1.1 Required Interface Adherence

This component must implement the following interface definition. Other components will be designed using this definition, so adherence to it is very important.

```
import java.util.Date;

public interface Workdays {

/**
* Constant value used to represent minutes as the unit of time.
*/
public static final int MINUTES = 0;

/**
* Constant value used to represent hours as the unit of time.
*/
public static final int HOURS = 1;

/**
* Constant value used to represent days as the unit of time.
*/
public static final int DAYS = 2;

/**
* Adds a non-workday to the list of non-work days
*
* @param nonWorkDay the date to add as a non work day
*/
public void addNonWorkday(Date nonWorkDay);

/**
* Removes a non-workday from the list of non-work days
*
* @param nonWorkDay the date to remove from the list
*/
public void removeNonWorkday(Date nonWorkDay);

/**
* Sets whether or not Saturday is to be considered a work day
*
* @param isSaturdayWorkday <code>true</code> if Saturday is to be
* considered a workday
*/
public void setSaturdayWorkday(boolean isSaturdayWorkday);

/**
* Returns whether or not Saturday is considered a workday.
*
* @returns <code>true</code> if Saturday is considered a workday.
```

```
*/
public boolean isSaturdayWorkday();

/**
* Sets whether or not Sunday is to be considered a work day
*
* @param isSundayWorkday <code>true</code> if Sunday is to be
* considered a workday
*/
public void setSundayWorkday(boolean isSundayWorkday);

/**
* Returns whether or not Sunday is considered a workday.
*
* @returns <code>true</code> if Sunday is considered a workday.
*/
public boolean isSundayWorkday();


/**
* Sets the hours of the workday start time.  This is to be in 24 hour
* mode.
*
* @param startTimeHours the hours of the workday start time
* (24 hour mode).
*/
public void setWorkdayStartTimeHours(int startTimeHours);

/**
* Returns the hours of the workday start time, in 24 hour mode.
*
* @returns the hours of the workday start time
*/
public int getWorkdayStartTimeHours();

/**
* Sets the minutes of the workday start time.
*
* @param startTimeMinutes the minutes of the workday start time
*/
public void setWorkdayStartTimeMinutes(int startTimeMinutes);

/**
* Returns the minutes of the workday start time.
*
* @returns the minutes of the workday start time
*/
public int getWorkdayStartTimeMinutes();

/**
* Sets the hours of the workday end time.  This is to be in 24 hour
* mode.
*
* @param endTimeHours the hours of the workday end time
* (24 hour mode).
*/
public void setWorkdayEndTimeHours(int endTimeHours);

/**
* Returns the hours of the workday end time, in 24 hour mode.
*
* @returns the hours of the workday end time
*/
public int getWorkdayEndTimeHours();
```

```
/**
* Sets the minutes of the workday end time.
*
* @param endTimeMinutes the minutes of the workday end time
*/
public void setWorkdayEndTimeMinutes(int endTimeMinutes);

/**
* Returns the minutes of the workday end time.
*
* @returns the minutes of the workday end time
*/
public int getWorkdayEndTimeMinutes();

/**
* Method to add a certain amount of time to a Date to caclulate the number
* of work days that it would take to complete.
*
* @param startDate the date to perform the addition to
* @param field the unit of time to add (minutes, hours, days)
* @param amount the amount of time to add
* @returns Date the result of adding the amount of time to the startDate
* taking into consideration the workdays definition.
*/
public Date add(Date startDate, int field, int amount);

}
```

2.1.2 Graphical User Interface Requirements

None.

2.1.3 External Interfaces

None.

2.1.4 Environment Requirements

- Development language: Java 1.5
- Compile target: Java 1.5, Java 1.6

2.1.4 Package Structure

com.topcoder.date.workdays

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

- The non-workdays, weekend day preferences and workday start and end times need to be configurable via the Configuration Manager.

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
- Base Exception 2.0
- Configuration API 1.0
- Configuration Persistence 1.0.2

**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:

Any third party library needs to be approved.

3.2.4 QA Environment:

- Java 1.5
- RedHat Linux 4
- 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 TC UML Tool.