Workdays 1.1 Component Specification

1. Design

All changes performed when synchronizing documentation with the version 1.0.1 of the source code of this component are marked with **purple**.

All changes made in the version 1.1 are marked with **blue**.

All new items in the version 1.1 are marked with **red**.

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 component provides a way to set holidays and other non-work days and to set whether weekend days are to be included as a normal workday. It also provides the ability to set the start and end hours and minutes of a workday.
The component has a 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 (the date returned is the end date). Similarly subtraction from the specific date can be performed with this component.
It also has the ability to load the initial non-workdays, weekend preferences, workday start time, and workday end time from a configuration file using the Configuration Manager component or Configuration API & Configuration Persistence components.
The configuration file may also contain the locale information (language, country, variant) – this is for internationalization and the style format of the dates (SHORT, MEDIUM, LONG and FULL), so the user may specify the dates in the configuration file using the local format and the style of the format he wants.
The configuration file can have 2 formats: properties and XML. Below are files supported when Configuration Manager is used:

- **properties**; the file may look like this:

```
ListDelimiter = %
startTime.hours = 8
startTime.minutes = 0
endTime.hours = 16
endTime.minutes = 30
isSaturdayWorkday = true
isSundayWorkday = false
locale.language = US
locale.country = us
locale.variant = MAC
dateStyle = SHORT
nonWorkdays = 3.23.2004%5.6.2004%7.2.2004%4.12.2004%6.18.2004
```

- **XML**; the file may look like this

```
<CMConfig>
  <Property name="startTime.hours">
    <Value>8</Value>
  </Property>
  <Property name="startTime.minutes">
    <Value>0</Value>
  </Property>
  <Property name="endTime.hours">
    <Value>16</Value>
  </Property>
  <Property name="endTime.minutes">
    <Value>30</Value>
  </Property>
  <Property name="isSaturdayWorkday">
    <Value>true</Value>
  </Property>
```

```
  <Property name="isSundayWorkday">
    <Value>false</Value>
  </Property>
  <Property name="locale.language">
    <Value>US</Value>
  </Property>
  <Property name="locale.country">
    <Value>us</Value>
  </Property>
  <Property name="locale.variant">
    <Value>MAC</Value>
  </Property>
  <Property name="dateStyle">
    <Value>SHORT</Value>
  </Property>
  <Property name="nonWorkdays">
    <Value>3.23.2004</Value>
    <Value>5.6.2004</Value>
    <Value>7.2.2004</Value>
    <Value>4.12.2004</Value>
    <Value>6.18.2004</Value>
  </Property>
</CMConfig>
```

And below are files supported when Configuration Persistence is used:

- **properties**; the file may look like this:

```
ListDelimiter = %
startTimeHours = 8
startTimeMinutes = 0
endTimeHours = 16
endTimeMinutes = 30
isSaturdayWorkday = true
isSundayWorkday = false
localeLanguage = US
localeCountry = us
localeVariant = MAC
dateStyle = SHORT
nonWorkdays = 3.23.2004%5.6.2004%7.2.2004%4.12.2004%6.18.2004
```

- **XML**; the file may look like this

```
<CMConfig>
  <Property name="startTimeHours">
    <Value>8</Value>
  </Property>
  <Property name="startTimeMinutes">
    <Value>0</Value>
  </Property>
  <Property name="endTimeHours">
    <Value>16</Value>
  </Property>
  <Property name="endTimeMinutes">
    <Value>30</Value>
  </Property>
  <Property name="isSaturdayWorkday">
    <Value>true</Value>
  </Property>
  <Property name="isSundayWorkday">
    <Value>false</Value>
  </Property>
  <Property name="localeLanguage">
```

```
    <Value>US</Value>
  </Property>
  <Property name="localeCountry">
    <Value>us</Value>
  </Property>
  <Property name="localeVariant">
    <Value>MAC</Value>
  </Property>
  <Property name="dateStyle">
    <Value>SHORT</Value>
  </Property>
  <Property name="nonWorkdays">
    <Value>3.23.2004</Value>
    <Value>5.6.2004</Value>
    <Value>7.2.2004</Value>
    <Value>4.12.2004</Value>
    <Value>6.18.2004</Value>
  </Property>
</CMConfig>
```

Locale information may be missing; in this case the default locale is used. If the style format is missing, the default format for the locale is used to parse the date strings.
For more information about the Locale and the DateFormat classes, read the javadocs for java.util.Locale and java.util.DateFormat.

In the version 1.1 this component was updated to support negative amount of time units to be added (i.e. subtraction of specific amount of time units). Also starting from the version 1.1 this component supports two configuration modes: via Configuration Manager component and via Configuration API & Configuration Persistence components. Usage of Configuration Manager starting from this version is deprecated. Additionally the source code was updated to use Java 1.5 language (generic parameters are now specified for collections).

1.1 Design Patterns

Abstract factory pattern – WorkdaysFactory interface represents a factory for Workdays implementations.
Strategy pattern – Workdays and WorkdaysFactory interfaces together with their implementations defined in this component can be used in some external strategy context.

1.2 Industry Standards

XML

1.3 Required Algorithms

The algorithm for DefaultWorkdays.add (startDate:Date, unitOfTime:WorkdaysUnitOfTime, amount:int) : Date.
The algorithm uses a few private methods to make it shorter and thus more readable.
First, let's consider the case when amount argument is positive.
At the beginning this algorithm transforms the amount to be added in minutes, to avoid a lot of if instructions in the algorithm. To do this, it uses a private method: *getAmountInMinutes (unitOfTime,amount):int* which is trivial.
Then it calculates the amount that needs to be added to the date corresponding to the beginning of the workday so the resulting date stays the same: adding 5 minutes to 08:03 has the same result as adding 8 minutes to 08:00 (this is a trivial example, but the rule is valid for any example).
Then it calculates the workday duration in minutes using *getWorkdayDurationInMinutes():int*, which is trivial.
Then, the amount of minutes that needs to be added are expressed in days (the days are, actually, workdays – they have the duration of a workday), hours and minutes. The hours and minutes are used only at the end of the algorithm. The days are used to determine the end date: we presume that all the days following the start date are normal days and we calculate the end date adding the number of days to the start date.
Then, the most important part takes place. Using a binary search we calculate, using

getWorkdaysCount(startCal, endCal, startDayIncluded):int, the number of non-workdays between the start date (inclusive) and the end date (exclusive). We add the number of days to the endDate, and repeat the cycle until the number of workdays is equal to the number of days to be added.
At the end we add the hours and minutes that we have determined and we have the end date we wanted.

In the version 1.1 this algorithm was updated to support negative input time amounts (i.e. the specified time amount is subtracted from the specific start date). The logic used for subtraction is very similar to the one described above. The only differences are listed below:

- When adjusting startDate and amount, the end time of the workday should be used instead of the start time.
- The adjusted startDate is used as maximum value in the binary search, not minimum.
- In the binary search start date is exclusive and end date is inclusive, but not vice versa.
- In the binary search days are subtracted from startDate, not added.
- When adjusting hours and minutes at the end of the algorithm, time should be subtracted from the value found by the binary search, not added.

Please see additional details for the main algorithm and all sub-routines mentioned above in method docs provided in TCUML.

1.4 Component Class Overview

DefaultWorkdays

The DefaultWorkdays class is the default implementation of the Workdays interface. It has a parameter less constructor to create an empty DefaultWorkdays instance, with default settings, and a constructor with a fileName:String and a fileFormat:String. The second constructor creates a DefaultWorkdays that loads its information from the file with the given fileName, of the given fileFormat. It provides methods for refreshing the configuration from the configuration file and for saving the modified configuration to the configuration file.

Changes in 1.1:
- Added support of additional configuration strategy with use of Configuration API and Configuration Persistence components.
- Usage of Configuration Manager for reading configuration is now deprecated.
- Added generic parameters for Java collection types as a part of code upgrade from Java 1.4 to Java 1.5.
- Added support of time subtraction (adding of negative amount) in add() method.

DefaultWorkdaysFactory

This is the default implementation of WorkdaysFactory. It creates instances of DefaultWorkdays type. It loads the configuration file under the constant namespace. If no error occurs and the namespace contains 2 properties:

file_name and file_format, it will create a DefaultWorkdays using the constructor with 2 string arguments. If the file_format is missing, the DefaultWorkdays.XML_FILE_FORMAT is used.

Changes in 1.1:
- Added support of additional configuration strategy with use of Configuration API and Configuration Persistence components.
- Usage of Configuration Manager for reading configuration is now deprecated.

Workdays [interface]

The Workdays Interface provides a set of generic functions to define a workday schedule (set holidays and other non-work days, set whether or not weekend days are to be included as a normal workday, set the start and end hours and minutes of a work day: for example, work day starts at 8:00AM and ends at 5:30PM) and a 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).

Changes in 1.1:
- Added generic parameters for Java collection types as a part of code upgrade from

Java 1.4 to Java 1.5.
- add() method should allow negative amount arguments.

WorkdaysFactory [interface]

This is the interface that every Workdays factory should implement.

An instance of this interface creates a Workdays instance.

WorkdaysUnitOfTime

Instances of this class represent units of time. It has a private constructor, so only the three public static final instances can be used: MINUTES, HOURS and DAYS.

Changes in 1.1:
- Trivial code fixes: Made name and value attributes final.

1.5 Component Exception Definitions

ConfigurationFileException

This exception is thrown if anything goes wrong in the process of loading the configuration file. So this exception is thrown:
-if the file doesn't exist;
-if the file can't be read: SecurityManager.checkRead(java.lang.String);
-if the configuration file is not well formed: the configuration manager throws an exception, or any other abnormal situation occurs;
-if the values in the configuration file are not in the required format (dates, hours, minutes)

Changes in 1.1:
- Made this class extend BaseCriticalException instead of BaseException.
- Added constructors that are required according to the current TopCoder standards.

1.6 Thread Safety

This component does not provide thread safety for the application using it. The application has to provide means to synchronize access to an instance of DefaultWorkdays, or access to the configuration file, if multiple threads are accessing it. However, if the Workdays is set and no modifications will take place, the add(…) method is thread safe, so it can be used by multiple threads.
Thread safety of this component was not changed in the version 1.1.

2. Environment Requirements

2.1 Environment

- Java 1.5 or higher.

2.2 TopCoder Software Components

- **Configuration Manager 2.1.5** – is used by DefaultWorkdaysFactory and DefaultWorkdays for loading and saving configuration to a file.
- **Base Exception 2.0** – is used by the custom exception defined in this component.
- **Type Safe Enum 1.1** – defines a base class for WorkdaysUnitOfTime.
- **Configuration API 1.0** – is used by DefaultWorkdaysFactory and DefaultWorkdays for reading/writing configuration parameters.
- **Configuration Persistence 1.0.2** – is used by DefaultWorkdaysFactory and DefaultWorkdays for reading/writing configuration files.

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

None

3. Installation and Configuration

3.1 *Package Name*

com.topcoder.date.workdays

3.2 *Configuration Parameters*

3.2.1 *Configuration of DefaultWorkdaysFactory*

DefaultWorkdaysFactory read its configuration from a predefined namespace ("com.topcoder.date.workdays") or file ("com/topcoder/date/workdays/defaultWorkdaysFactory.properties") with use of Configuration Manager or Configuration Persistence component. In case if Configuration API / Configuration Persistence are used (useConfigurationAPI == true), "file_format" parameter is not used since Configuration Persistence automatically detects file type by file name extension. DefaultWorkdaysFactory supports the following parameters:

Parameter	Description	Values
file_name	The configuration file name for DefaultWorkdays. Note that when DefaultWorkdaysFactory is configured with use of Configuration Manager, DefaultWorkdays instances generated by it are also configured with use of Configuration Manager; and vice versa.	String. Not empty. Required.
file_format	The format of the configuration file for DefaultWorkdays. Is ignored when Configuration API/Configuration Persistence is used. Default is "XML".	"XML" or "properties". Optional.

3.2.2 *Configuration of DefaultWorkdays*

DefaultWorkdays read its configuration from the specified file that is read with use of Configuration Manager (CM) or Configuration Persistence (CP) component. DefaultWorkdays supports the following parameters:

Parameter	Description	Values
isSaturdayWorkday	The value indicating whether Saturdays are to be considered as a normal workdays or not.	"true" or "false" (case insensitive). Required.
isSundayWorkday	The value indicating whether Sundays are to be considered as a normal workdays or not.	"true" or "false" (case insensitive). Required.
nonWorkdays	The list of non-workdays (e.g. holidays).	String[]. Each element should be a String representation of Date.
startTime.hours for CM startTimeHours for CP	The workdays start time hour.	String representation of integer in the range [0, 24]. Required.
startTime.minutes for CM startTimeMinutes for CP	The workdays start time minute. Should be "0" if startTime.hours is equal to "24".	String representation of integer in the range [0, 59]. Required.

endTime.hours for CM endTimeHours for CP	The workdays end time hour.	String representation of integer in the range [0, 24]. Required.
endTime.minutes for CM endTimeMinutes for CP	The workdays end time minute. Should be “0” if endTime.hours is equal to “24”.	String representation of integer in the range [0, 59]. Required.
locale.language for CM localeLanguage for CP	The language part of locale to be used when parsing/formatting dates. If not specified, the default locale is used.	String. Optional.
locale.country for CM localeCountry for CP	The country part of locale to be used when parsing/formatting dates. If not specified, the default locale is used.	String. Optional.
locale.variant for CM localeVariant for CP	The variant part of locale to be used when parsing/formatting dates. If not specified, the locale is created without a variant.	String. Optional.
dateStyle	The date style to be used for string representation of dates provided in nonWorkdays parameter. If not specified, default date format is used for the specified locale.	“SHORT”, “MEDIUM”, “LONG” or “FULL” (case insensitive). Optional.

Note that time “endTime.hours:endTime.minutes” should be greater than “startTime.hours: startTime.minutes”.

3.3 Dependencies Configuration

None

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

Your running application might do this:

- create a DefaultWorkdays, passing to the constructor the name of a configuration file
- use the instance as a Workdays object and perform the available operations

4.3 Demo

4.3.1 Sample configuration for Configuration Manager

./sample.properties

```
ListDelimiter = %
startTime.hours = 8
startTime.minutes = 0
endTime.hours = 16
endTime.minutes = 0
isSaturdayWorkday = true
isSundayWorkday = false
locale.language = US
locale.country = us
```

```
dateStyle = SHORT
nonWorkdays = 12.13.2010%12.15.2010%12.25.2010
```

4.3.2 API usage sample

```
// create an instance of DefaultWorkdays using Configuration API / Configuration Persistence
String configFile = "./sample.properties";
DefaultWorkdays workdays = new DefaultWorkdays(configFile);

// create an instance of DefaultWorkdays using Configuration Manager
// Note that this constructor is now deprecated
workdays = new DefaultWorkdays(configFile, DefaultWorkdays.PROPERTIES_FILE_FORMAT);

// add a non-work day
workdays.addNonWorkday(DateFormat.getDateInstance().parse("1.7.2005"));

// get all configured non-workdays
Set<Date> nonWorkdays = workdays.getNonWorkdays();
// nonWorkdays must contain a single element equal to
//     DateFormat.getDateInstance().parse("1.7.2005")

// remove a non-work day
workdays.removeNonWorkday(DateFormat.getDateInstance().parse("1.7.2005"));

// set weekend days to be normal work-days
workdays.setSaturdayWorkday(true);
workdays.setSundayWorkday(true);

// query if weekend days are normal work-days
boolean isSaturdayWorkday = workdays.isSaturdayWorkday();
boolean isSundayWorkday = workdays.isSundayWorkday();

// change the start and end time hours and minutes
workdays.setWorkdayStartTimeHours(8);
workdays.setWorkdayStartTimeMinutes(0);
workdays.setWorkdayEndTimeHours(16);
workdays.setWorkdayEndTimeMinutes(30);

// query the start and end time hours and minutes
int startTimeHours = workdays.getWorkdayStartTimeHours();
int startTimeMinutes = workdays.getWorkdayStartTimeMinutes();
int endTimeHours = workdays.getWorkdayEndTimeHours();
int endTimeMinutes = workdays.getWorkdayEndTimeMinutes();

// calculate the end date, knowing the start date, the amount of time and the unit of time
Date startDate = new Date();
Date endDate = workdays.add(startDate, WorkdaysUnitOfTime.HOURS, 40);

// calculate the end date by subtracting 40 hours for the specified start date
endDate = workdays.add(startDate, WorkdaysUnitOfTime.HOURS, -40);

// refresh the configuration from the file
workdays.refresh();

// change the file name
String fileName = ...
workdays.setFileName(fileName);

// change the file format to XML
// Note that this method is now deprecated
workdays.setFileFormat(DefaultWorkdays.XML_FILE_FORMAT);

// save the changes made by the program to the configuration file
workdays.save();

// create a Workdays factory using Configuration Manager
// Note that this constructor is now deprecated
WorkdaysFactory factory = new DefaultWorkdaysFactory();

// create a Workdays factory using Configuration API / Configuration Persistence
factory = new DefaultWorkdaysFactory(true);

// use the Workdays factory to create instances of Workdays
Workdays w1 = factory.createWorkdaysInstance();
Workdays w2 = factory.createWorkdaysInstance();
```

4.3.3 Sample calculations of adding/subtracting time from a date

Assume that configuration from the section 4.3.1 is used.
Assume that startDate passed to DefaultWorkdays#add() method is equal to ***2010-12-14 14:20***.
The calendar below shows workdays and non-workdays (marked with yellow) in this month:

Sun	Mon	Tue	Wed	Thu	Fri	Sat
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Then assuming that unitOfTime = WorkdaysUnitOfTime.HOURS and amount = 38, the value returned by the add() method must be ***2010-12-21 12:20***.

And assuming that unitOfTime = WorkdaysUnitOfTime.HOURS and amount = -38 (negative), the value returned by the add() method must be ***2010-12-08 8:20***.

5. Future Enhancements

- Different implementations of the Workdays interface, with different persistence support.
- The workdays interface could have more methods.