Workday Calendar documentation

Table of Contents

Workday Calendar is a customizable module for workday calculations using in-memory data. After initialization, the module provides methods for querying the n-th workday before or after a given start date and querying a list of workdays between two dates.

1. Installation

Min. JDK version: 1.8

This project is available in maven central as:

```
<dependency>
    <groupId>hu.icellmobilsoft.wdc</groupId>
    <artifactId>wdc-calculator-core-action</artifactId>
    <version>1.2.0</version>
```

```
</dependency>
```

2. Initialization, configurations

Workday Calendar uses an in-memory cache to store the necessary calendar data. The cache initializes as follows:

1. After launching the application, the cache gets initialized with data of three years: the actual year, last year and next year. This step uses in-built Java functions to determine whether a given date is a workday or weekend day.
2. After the first step, the module uses Microprofile Config to obtain the following three environmental variables: Data file folder, Include Days and Exclude Days.
3. Then, files under Data File Folder variable get processed. See Data File Folder
4. Next, dates given in Include Days variable get processed. See Include Days
5. Finally, dates given in Exclude Days variable get processed. See Exclude Days

2.1. Data File Folder

The data file folder config key is "wdc.config.input.datafolder". This variable determines the path for input files. The purpose of these files is to add holidays and working weekends to the cache.

Files are expected to be in .xml format and contain dates and workday flags (example: datafile.xml). Every file is validated against workdayData.xsd. For every date entry in a file, the corresponding workday flag gets overwritten in the cache.

If a given date is not present in the cache yet, its year gets initialized as in the first step of the cache initialization process.

2.2. Include Days

The include days config key is "wdc.config.input.include". This variable should contain dates separated with semicolons (example: 2019-12-07;2019-12-14). These determine workday dates, that should overwrite previous values.

If a given date is not present in the cache yet, its year gets initialized as in the first step of the cache initialization process.

2.3. Exclude Days

The exclude days config key is "wdc.config.input.exclude". This variable should contain dates separated with semicolons (example: 2019-12-07;2019-12-14). These determine non-workday dates, that should overwrite previous values.

If a given date is not present in the cache yet, its year gets initialized as in the first step of the cache initialization process.

3. Usage

The workday calculation methods are accessible through CalculatorCoreAction class. The available methods are explained below.

3.1. calculateWorkday() method

The calculateWorkday() method is used for querying the n-th workday before or after a given date.

Input:

- LocalDate startDate: must be between years 1 and 2100
- int numberOfWorkdays: positive or negative integer, can’t be zero

Output:

- LocalDate: startDate + numberOfWorkdays

Exceptions:

- BusinessException on invalid input parameters

Example:

```
Given we have in cache:
(2020, 5, 15) - workday
(2020, 5, 16) - not workday
(2020, 5, 17) - not workday
(2020, 5, 18) - workday
(2020, 5, 19) - workday
(2020, 5, 20) - workday
Then:
calculateWorkday(LocalDate.of(2020, 5, 15), 3) = (2020, 5, 20)
```

3.2. calculateWorkdayList() method

The calculateWorkdayList() method is used for querying a list of workdays between two dates.

Input:

- LocalDate startDate: must be between years 1 and 2100
- LocalDate endDate: must be between years 1 and 2100, can’t be earlier than startDate

Output:

- List<LocalDate>: workday dates between startDate and endDate (both inclusive)

Exceptions:

- BusinessException on invalid input parameters

Example:

```
Given we have in cache:
(2020, 5, 9) - not workday
(2020, 5, 10) - not workday
(2020, 5, 11) - workday
(2020, 5, 12) - workday
(2020, 5, 13) - workday
(2020, 5, 14) - workday
(2020, 5, 15) - workday
Then:
calculateWorkdayList(LocalDate.of(2020, 5, 9), LocalDate.of(2020, 5, 15)) = [
(2020, 5, 11), (2020, 5, 12), (2020, 5, 13), (2020, 5, 14), (2020, 5, 15)]
```

3.3. isGuaranteedResultOfCalculateWorkday() method (from v1.2.0)

Returns whether the result of the given calculateWorkday() method is guaranteed.

Input:

- LocalDate startDate: must be between years 1 and 2100
- int numberOfWorkdays: positive or negative integer, can’t be zero

Output:

- boolean: returns true if the processing only affects the years of the source file, otherwise it returns false

Exceptions:

- BusinessException on invalid input parameters

Example:

```
Given source file contains data only year 2020.
Then:
isGuaranteedResultOfCalculateWorkday(LocalDate.of(2020, 12, 31), 1) = false
```

3.4. isGuaranteedResultOfCalculateWorkdayList() method (from v1.2.0)

Returns whether the result of the given calculateWorkdayList() method is guaranteed.

Input:

- LocalDate startDate: must be between years 1 and 2100
- LocalDate endDate: must be between years 1 and 2100, can’t be earlier than startDate

Output:

- boolean: returns true if the processing only affects the years of the source file, otherwise it returns false

Exceptions:

- BusinessException on invalid input parameters

Example:

```
Given source file contains data only year 2020.
Then:
isGuaranteedResultOfCalculateWorkdayList(LocalDate.of(2020, 12, 1), LocalDate.of(2021,
1, 1)) = false
```

4. Implementation

A simple CDI based implementation of the Workday Calendar module would look like this:

CalculatorExample.java

```
package somepackage;

import java.time.LocalDate;
import javax.inject.Inject;
import hu.icellmobilsoft.wdc.calculator.core.action.CalculatorCoreAction;

public class CalculatorExample {

    @Inject
    private CalculatorCoreAction calculatorCoreAction;

    public void callMethods() {
        private LocalDate startDate = LocalDate.of(2020, 5, 9);
        private LocalDate endDate = LocalDate.of(2020, 5, 15);
        private int days = 3;
        try {
            LocalDate workday = calculatorCoreAction.calculateWorkday(startDate,
numberOfWorkdays);
            List<Workday> workdayList =
calculatorCoreAction.calculateWorkdayList(startDate, endDate);
        } catch (BusinessException e) {
            doSomething();
        }
    }
}
```

5. Migration description

5.1. v1.1.0 → v1.2.0

Workday Calendar v1.1.0→v1.2.0 migration descriptions, new features, changes.

5.1.1. Indication of relocated holidays

This feature allows you to mark working days as a holiday and vice versa. It is possible to define type, replacement day and description for the calendar days in the source file. Possible values for the type (HolidayType): FEASTDAY, WEEKEND, RESTDAY, PLUSWORKINGDAY, NATIONALDAY.

workdayData.xsd changes

WorkdayDataType has been expanded with optional fields

```
<xsd:complexType name="WorkdayDataType">
        <xsd:annotation>
            <xsd:documentation xml:lang="hu">Munkanap jelzésére szolgáló komplex
adattípus.</xsd:documentation>
            <xsd:documentation xml:lang="en">Complex type for indicating
workday.</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            ....
            <xsd:element name="holidayType" type="HolidayType" minOccurs="0">
                <xsd:annotation>
                    <xsd:documentation xml:lang="hu">Munka, vagy munkaszüneti nap
típusa.</xsd:documentation>
                    <xsd:documentation xml:lang="en">Type of workday or
holiday.</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="substitutedDay" type="common:DayType" minOccurs="0">
                <xsd:annotation>
                    <xsd:documentation xml:lang="hu">Kiváltott naptári
nap.</xsd:documentation>
                    <xsd:documentation xml:lang="en">Substituted calendar
day.</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="description" type="xsd:string" minOccurs="0">
                <xsd:annotation>
                    <xsd:documentation xml:lang="hu">Leírása a
napnak.</xsd:documentation>
                    <xsd:documentation xml:lang="en">Description of the
day.</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
```

Source file example to define new optional data

```
<workdayData>
    <date>2020-08-21</date>
    <workday>0</workday>
    <holidayType>RESTDAY</holidayType>
    <substitutedDay>2020-08-29</substitutedDay>
    <description>moved Friday after 20th of August</description>
</workdayData>
<workdayData>
    <date>2020-08-29</date>
    <workday>1</workday>
    <holidayType>PLUSWORKINGDAY</holidayType>
    <substitutedDay>2020-08-21</substitutedDay>
    <description>instead of 20th of August</description>
</workdayData>
```

Change-over

The changes do not result in migration work, they are backward compatible.

5.1.2. Warning for uninitialized data

Changes

- min and max constant for the years (min: 1) max (2100)
- if the processing concerns a year that does not fall within this range than exception occurs
- the initialization for 3 years (current, before and after the given year) remains at the start of the module, but if it is required during execution and the cache does not yet contain the affected year then initialize it (weekday is workday, weekend is non-working day)
- the years specified in the input file are considered reliable years, if the processing concerns other years then it is no longer reliable
- two new methods have been implemented to decide whether processing only affects reliable (guaranteed) or unreliable years:

```
/**
     * Returns whether the result of the given calculateWorkday method is guaranteed.
     *
     * @param startDate
     *            From date.\n Valid dates between years 1 and 2100.
     * @param numberOfWorkdays
     *            Number of workdays.\n Cannot be 0, may be negative.
     * @return boolean
     * @throws BusinessException
     *             Throws on invalid input parameters
     */
    public boolean isGuaranteedResultOfCalculateWorkday(LocalDate startDate, int
numberOfWorkdays) throws BusinessException

    /**
     * Returns whether the result of the given calculateWorkdayList method is
guaranteed.
     *
     * @param startDate
     *            From date (inclusive).\n Valid dates between years 1 and 2100.
     * @param endDate
     *            To date (inclusive).\n Valid dates between years 1 and 2100.
     * @return boolean
     * @throws BusinessException
     *             Throws on invalid input parameters
     */
    public boolean isGuaranteedResultOfCalculateWorkdayList(LocalDate startDate,
LocalDate endDate) throws BusinessException
```

Change-over

The changes do not result in migration work, they are backward compatible.

5.1.3. Source folder synchronization

This feature allows you to not have to restart the module after modifying the source data file. The source folder is checked at regular intervals (hourly), and if the contents of the folder have changed, the module will re-initialize itself.

Change-over

The changes do not result in migration work, they are backward compatible.

5.1.4. Java11 upgrade

JDK version upgraded 1.8 to 1.11

Change-over

Workday Calendar can only be used where the JDK version is at least 1.11.

5.1.5. Replacing project-specific Exception with Coffee Exception

The project-specific BusinessException has been removed and instead the CalculatorCoreAction methods will throw the Coffee BusinessException if any validation is failed.

Change-over

You must already expect the Coffee BusinessException when using the CalculatorCoreAction methods.

5.2. v1.2.0 → v1.3.0

Workday Calendar v1.2.0→v1.3.0 migration descriptions, new features, changes.

5.2.1. JavaEE 10 support

Added the following module alternatives which supports JavaEE 10:

- wdc-calculator-core-dto-jee-10
- wdc-calculator-core-action-jee-10

Change-over

The changes do not result in migration work, they are backward compatible.

5.2.2. parent pom upgrade

The parent pom is upgraded from 1.1.0 to 1.4.0, in order to support JDK 17 compilation.

Change-over

The changes do not result in migration work, they are backward compatible.