Red Hat Decision Manager 7.4

Running and modifying the employee roster starter application for Red Hat Business Optimizer using an IDE

Last Updated: 2020-05-26

Red Hat Decision Manager 7.4 Running and modifying the employee roster starter application for Red Hat Business Optimizer using an IDE

Red Hat Customer Content Services
brms-docs@redhat.com

Legal Notice

Abstract

This document describes how to run and modify the OptaShift Employee Rostering template included as an add-on in Red Hat Decision Manager 7.4.

Table of Contents

PREFACE

As a business rules developer, you can use an IDE to build, run, and modify the **employee-rostering** starter application that uses the Red Hat Business Optimizer functionality.

Prerequisites

- You use an integrated development environment, such as Eclipse (including Red Hat Developer Studio) or IntelliJ IDEA.
- You have an understanding of the Java language.

CHAPTER 1. OVERVIEW OF THE EMPLOYEE ROSTERING STARTER APPLICATION

The employee rostering starter application assigns employees to shifts on various positions in an organization. For example, you can use the application to distribute shifts in a hospital between nurses, guard duty shifts across a number of locations, or shifts on an assembly line between workers.

Optimal employee rostering must take a number of variables into account. For example, different skills can be required for shifts in different positions. Also, some employees might be unavailable for some time slots or might prefer a particular time slot. Moreover, an employee can have a contract that limits the number of hours that the employee can work in a single time period.

The Red Hat Business Optimizer rules for this starter application use both hard and soft constraints. During an optimization, the planning engine may not violate hard constraints, for example, if an employee is unavailable (out sick), or that an employee cannot work two spots in a single shift. The planning engine tries to adhere to soft constraints, such as an employee's preference to not work a specific shift, but can violate them if the optimal solution requires it.

CHAPTER 2. BUILDING AND RUNNING THE EMPLOYEE ROSTERING STARTER APPLICATION

You can build the employee rostering starter application from the source code and run it using a JBoss EAP or WildFly application server.

You can use the command line to build the application, then install it into a stand-alone server.

Alternatively, you can use your IDE, for example, Eclipse (including Red Hat JBoss Developer Studio), to build and run the application.

You can also deploy and run the pre-built WAR file that is supplied with the source code.

For information about using the application, see Deploying and using the employee roster starter application for Red Hat Business Optimizer on Red Hat OpenShift Container Platform.

2.1. PREPARING DEPLOYMENT FILES

You must download and prepare the deployment files before building and deploying the application.

Procedure

1. Download the **rhdm-7.4.0-add-ons.zip** file from the Software Downloads page for Red Hat Decision Manager 7.4.
2. Unzip the downloaded archive.
3. Unzip the **rhdm-7.4.0-employee-rostering.zip** file that is extracted from the add-ons archive. The **employee-rostering-distribution-7.23.0.Final-redhat-00002** folder is created. This folder is the base folder in subsequent parts of this document.

NOTE

File and folder names might have higher version numbers than specifically noted in this document.

2.2. BUILDING AND RUNNING THE EMPLOYEE ROSTERING STARTER APPLICATION FROM THE COMMAND LINE

You can use the command line to build the employee rostering starter application and run it.

If you use this procedure, the data is stored in memory and is lost when the server is stopped. To build and run the application with a database server for persistent storage, see Section 2.3, “Building and running the employee rostering starter application with persistent data storage from the command line”.

Prerequisites

- You prepared the deployment files as described in Section 2.1, “Preparing deployment files”.
- A Java Development Kit is installed.
- Maven is installed.

- The host has access to the Internet. The build process uses the Internet for downloading Maven packages from external repositories.

Procedure

1. In a terminal window, change to the **sources** directory.
2. Run the following command:

```
mvn clean install
```

3. Wait for the build process to complete.
4. Use one of the following methods to run the application:
   a. Run the WildFly server deployed as part of the build process:
      i. In the **local/appserver/wildfly-14.0.1-final/standalone/deployments** subdirectory, create the **optaweb-employee-rostering-webapp-<version>.war.dodeploy** file. The **<version>** must be the same as in the existing **optaweb-employee-rostering-webapp-<version>.war** symlink in the same directory.
      ii. In the **local/appserver/wildfly-14.0.1-final/bin** subdirectory, run the **./standalone.sh** command.
   b. Deploy the **optaweb-employee-rostering-webapp/target/optaweb-employee-rostering-*.war** file into an existing WildFly or JBoss EAP server and start the application server.
   c. Use the following commands to run a server using Maven:

```
mvn -N wildfly:start wildfly:deploy
mvn gwt:codeserver
```

NOTE

If you use Maven to start the server, the UI uses the gwt codeserver and monitors the gwtui source. In this case, if you change gwtui code, the codeserver picks the changes up automatically; there is no need to rebuild the application for gwtui code changes.

Later, to stop this server, use the command:

```
mvn -N wildfly:shutdown
```

5. To access the application, enter http://localhost:8080/gwtui/gwtui.html in a web browser.

2.3. BUILDING AND RUNNING THE EMPLOYEE ROSTERING STARTER APPLICATION WITH PERSISTENT DATA STORAGE FROM THE COMMAND LINE

If you use the command line to build the employee rostering starter application and run it, you can provide a database server for persistent data storage.

Prerequisites

Prerequisites

- You prepared the deployment files as described in Section 2.1, "Preparing deployment files".
- A Java Development Kit is installed.
- Maven is installed.
- The host has access to the Internet. The build process uses the Internet for downloading Maven packages from external repositories.
- You have a deployed WildFly or Red Hat JBoss EAP application server and a deployed MySQL or PostrgeSQL database server.
- You have set up a JDBC data source in the application server for the database server.

Procedure

1. In a terminal window, change to the **sources** directory.
2. Run the following command:

```
mvn clean install -Dproductized -Dorg.optaweb.employeerostering.persistence.datasource=
<dsname> -Dorg.optaweb.employeerostering.persistence.dialect=<dialect>
```

   In the preceding command, replace the following values:

   - *<dsname>* with the name of the data source in the application server.
   - *<dialect>* with one of the following strings, depending on the type of database server:
     - For MySQL, **org.hibernate.dialect.MySQL5Dialect**
     - For PostgreSQL, **org.hibernate.dialect.PostgreSQLDialect**
3. Wait for the build process to complete.
4. Deploy the **optaweb-employee-rostering-webapp/target/optaweb-employee-rostering-7.23.0.Final-redhat-00002.war** directory into the application server and start the application server.
5. To access the application, enter http://localhost:8080/gwtui/gwtui.html in a web browser.

2.4. BUILDING AND RUNNING THE EMPLOYEE ROSTERING STARTER APPLICATION USING ECLIPSE

You can use Eclipse, including Red Hat JBoss Development Studio, to build the employee rostering starter application and run it.

Prerequisites

- You prepared the deployment files as described in Section 2.1, "Preparing deployment files".
- Eclipse is installed.

- The host has access to the Internet. The build process uses the Internet for downloading Maven packages from external repositories.
- Google Chrome is installed in order to run the application with the suggested configuration.

NOTE

You can modify the configuration to use another web browser.

Procedure

1. Start Eclipse.
2. From the main menu, select **File** > **Import...**.
3. Select the **Maven** > **Existing Maven projects** wizard.
4. For the root directory, select the root directory of the application source.
5. Click **Finish**.
6. Optionally, to avoid seeing many errors in Eclipse:
   a. In the root directory of the application source, run the **mvn clean install** command and wait for the build to finish.
   b. In the Eclipse navigation tree, right-click **employee-rostering-shared** and select **Build Path** > **Configure Build Path...**.
   c. Click the **Source** tab, then click **Add Folder...**.
   d. Select the **employee-rostering-shared/target/generated-sources** folder and click **OK**.
7. From the main menu, select **Run** > **External Tools** > **External Tools Configurations...**.
8. Under **Program**, create the following launch configurations:
   a. Open OptaWeb Employee Rostering in Chrome:
      - Name: **Open OptaWeb Employee Rostering in Chrome**
      - Location: **/usr/bin/google-chrome**
      - Working Directory: **${workspace_loc:/employee-rostering}**
      - Arguments: **--incognito http://localhost:8080/gwtui/gwtui.html**

NOTE

You can change the name, location. and arguments to use another browser instead of Chrome.

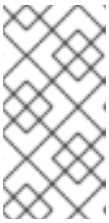
   b. Kill Code Server:
      - Name: **Kill Code Server**
      - Location: **/usr/sbin/fuser**

- Working Directory: **${workspace_loc:/employee-rostering}**
- Arguments: **fuser -k 9876/tcp**

9. From the main menu, select **Run** > **Run Configurations...**.
10. Under **Maven Build**, create the following launch configurations:
    a. OptaWeb Employee Rostering Build:
       - Name: **OptaWeb Employee Rostering Build**
       - Base directory: **${workspace_loc:/employee-rostering}**
       - Goals: **clean install**
       - Parameter: **gwt:skipCompilation** Value: **true**
    b. OptaWeb Employee Rostering Start Code Server:
       - Name: **OptaWeb Employee Rostering Start Code Server**
       - Base directory: **${workspace_loc:/employee-rostering}**
       - Goals: **gwt:codeserver**
       - Parameter: **gwt:skipCompilation** Value: **true**
    c. OptaWeb Employee Rostering Start Webserver:
       - Name: **OptaWeb Employee Rostering Start Webserver**
       - Base directory: **${workspace_loc:/employee-rostering}**
       - Goals: **wildfly:start wildfly:deploy**
       - Parameter: **gwt:skipCompilation** Value: **true**
    d. OptaWeb Employee Rostering Stop Webserver:
       - Name: **OptaWeb Employee Rostering Stop Webserver**
       - Base directory: **${workspace_loc:/employee-rostering}**
       - Goals: **wildfly:shutdown**
       - Parameter: **gwt:skipCompilation** Value: **true**
11. Under **Launch Group**, create a launch group named **OptaWeb Employee Rostering Run**. Add the following launches to it:
    - **Program::Kill Code Server** Launch mode: **inherit** Post launch action: **Wait until terminated**
    - **Maven Build::OptaWeb Employee Rostering Stop Webserver** Launch mode: **inherit** Post launch action: **Wait until terminated**
    - **Maven Build::OptaWeb Employee Rostering Build** Launch mode: **inherit** Post launch action: **Wait until terminated**

- **Maven Build::OptaWeb Employee Rostering Start Webserver** Launch mode: **inherit** Post launch action: none
- **Maven Build::OptaWeb Employee Rostering Start Code server** Launch mode: **inherit** Post launch action: **Wait for console output (regexp): The code server is ready at**
- **Program::Open OptaWeb Employee Rostering in Chrome** Launch mode: **inherit** Post launch action: none

12. To build, run, and immediately acccess the application, run the **OptaWeb Employee Rostering Run** launch group. You can then change the application and rerun the launch group to test your changes.



NOTE

When you use this method to run the application, the UI uses the gwt codeserver and monitors the gwtui source. If you change gwtui code, the codeserver picks the changes up automatically; there is no need to rebuild the application for gwtui code changes.

2.5. RUNNING THE SUPPLIED PRE-BUILT WAR FILE

To access the Employee Rostering starter application, you can deploy and run the pre-built **employee-rostering-webapp-7.23.0.Final-redhat-00002.war** file that is supplied with Red Hat Decision Manager.

Prerequisites

- You prepared the deployment files as described in Section 2.1, “Preparing deployment files”.
- A Java Development Kit is installed.
- You have a deployed WildFly or Red Hat JBoss EAP application server.

Procedure

1. In your WildFly or Red Hat JBoss EAP application server, enable property replacement. For more information about enabling and disabling property replacement in Red Hat JBoss EAP, see Property Replacement.
   a. Start the application server in standalone mode.
   b. Start the management CLI by entering the following command:

```
EAP_HOME/bin/jboss-cli.sh --connect
```

   c. In the management CLI enter the following command:

```
/subsystem=ee:write-attribute(name="spec-descriptor-property-replacement",value=true)
```

2. Stop the application server.
3. Deploy the **binaries/employee-rostering-webapp-7.23.0.Final-redhat-00002.war** directory into the application server and start the application server.
4. To access the application, enter http://localhost:8080/gwtui/gwtui.html in a web browser.

CHAPTER 3. OVERVIEW OF THE SOURCE CODE OF THE EMPLOYEE ROSTERING STARTER APPLICATION

The employee rostering starter application consists of the following principal components:

- The *server* that implements the rostering logic using Red Hat Business Optimizer and provides a REST API.
- The *client* that implements a user interface using the **gwt** library and calls the server using the REST API.

While some code is shared between the client and the server, you can build and use these components independently. In particular, you can implement a different user interface and use the REST API to call the server.

In addition to the two main components, the employee rostering template contains a generator of random source data (useful for demonstration and testing purposes) and a benchmarking application.

Modules and key classes

The Java source code of the employee rostering template contains several Maven modules. Each of these modules includes a separate Maven project file (**pom.xml**), but they are intended for building in a common project.

The modules contain a number of files, including Java classes. This document lists all the modules, as well as the classes and other files that contain the key information for the employee rostering calculations.

- **employee-rostering-benchmark** module: Contains an additional application that generates random data and benchmarks the solution.
- **employee-rostering-distribution** module: Contains readme files.
- **employee-rostering-docs** module: Contains documentation files.
- **employee-rostering-gwtui** module: Contains the client application with the user interface, developed using the **gwt** toolkit.
- **employee-rostering-server** module: Contains the server application that uses Red Hat Business Optimizer to perform the rostering calculation.
    - **src/main/resources/org/optaweb/employeerostering/server/solver/employeeRosteringScoreRules.drl** file: Contains the rules for the Red Hat Business Optimizer calculation. These rules are written in the Drools rules language. You can modify the rules to change the logic for employee rostering.
    - **src/main/java/org.optaweb.employeerostering.server.roster/rosterGenerator.java** class: generates random input data for demonstration and testing purposes. If you change the requires input data, change the generator accordingly.
- **employee-rostering-shared** module: Contains the data structures shared between the server and client applications. In particular, under **src/main/java/org/optaweb/employeerostering/shared/***, this module includes the Java classes that define the input data for the rostering calculations, including the following classes:
    - **employee/EmployeeAvailability.java** defines availability information for an employee. For every time slot, an employee can be unavailable, available, or it can be a preferred timeslot

for the employee.

- **employee/Employee.java** defines an employee. An employee has a name, a list of skills, and works under a contract. Skills are represented by **EmployeeSkillProficiency** objects.
- **roster/Roster.java** defines the calculated rostering information.
- **shift/Shift.java** defines a shift to which an employee can be assigned. A shift is defined by a time slot and a spot. For example, in a diner there could be a shift in the **Kitchen** spot for the **February 20 8AM-4PM** time slot. Multiple shifts can be defined for a given spot and time slot; in this case, multiple employees are required for this spot and time slot.
- **skill/Skill.java** defines a skill that an employee can have.
- **spot/Spot.java** defines a spot where employees can be placed. For example, in a diner **Kitchen** can be a spot.
- **contract/Contract.java** defines a contract that sets the limits of work time for an employee in various time periods.
- **tenant/Tenant.java** defines a tenant. Each tenant represents an independent set of data; any change in the data for one tenant does not affect any other tenants.
  The **employee-rostering-shared** module also includes other shared artifacts:
- ***View.java** classes define value sets that are calculated from other information; the client application can read these values through the REST API, but not write them.
- ***RestService.java** interfaces define the REST API. Both the server and the client application separately define implementations of these interfaces.

- **employee-rostering-shared-gwt** module: Contains some of the classes required for the GWT client application.
- **employee-rostering-webapp** module: Contains the HTML and other files necessary for building the entire application (client and server).

CHAPTER 4. MODIFYING THE EMPLOYEE ROSTERING STARTER APPLICATION

To modify the employee rostering starter application to suit your needs, you must change the rules that govern the optimization process. You must also ensure that the data structures include the required data and provide the required calculations for the rules. If the required data is not present in the user interface, you must also modify the user interface.

The following procedure outlines the general approach to modifying the employee rostering starter application.

Prerequisites

- You have a build environment that successfully builds the application.
- You can read and modify Java code.

Procedure

1. Plan the required changes. Answer the following questions:
    - What are the additional scenarios that *must* be avoided? These scenarios are *hard constraints*.
    - What are the additional scenarios that the optimizer must *try to avoid* when possible? These scenarios are *soft constraints*.
    - What data is required to calculate if each scenario is happening in a potential solution?
    - Which of the data can be derived from the information that the user enters in the existing version?
    - Which of the data can be hardcoded?
    - Which of the data must be entered by the user and is not entered in the current version?
2. If any required data can be calculated from the current data or can be hardcoded, add the calculations or hardcoding to existing view or utility classes. If the data must be calculated on the server side, add REST API endpoints to read it.
3. If any required data must be entered by the user, add the data to the classes representing the data entities (for example, the **Employee** class), add REST API endpoints to read and write the data, and modify the user interface to enter the data.
4. When all the data is available, modify the rules. For most modifications, you must add a new rule. The rules are located in the **src/main/resources/org/optaweb/employeerostering/server/solver/employeeRosteringScoreRules.drl** file of the **employee-rostering-server** module.
   Use the Drools language for the rules. For reference information about the Drools rule language, see Rule Language Reference. Classes defined in the **optaweb-employee-rostering-shared** and **optaweb-employee-rostering-server** modules are available to the decision engine.
5. After modifying the application, build and run it.

APPENDIX A. VERSIONING INFORMATION

Documentation last updated on Friday, May 22, 2020.