Table of Contents

Introduction 1
Requirements 1
Example 1
Configuration 2
Job Configuration 2
Datasource Configuration 3
Logging 4
Testing 5

Version: 1.0 / 2020-03-04

Introduction

This document describes the integration of Quartz Job Scheduler with VisionX workflow module. Basically, it explains how a workflow can be started by Quartz Job Scheduler.

Requirements

We need following tools and libraries:

- Eclipse IDE (EE variant)
- Tomcat application server, 8 or newer
- Quartz release, 2.3.0 or newer
- Javax transaction API (javax.transaction-api-1.3.jar), 1.3 or newer
- Java Mail API (javax.mail-1.5.4.jar), 1.4 or newer
- JDBC drivers for datasources used in config.xml
- wfengineserver.jar
- jvx.jar
- appsclient.jar

Example

First, create a new dynamic web project in Eclipse IDE. After that, add all Quartz libraries from the **quartz_install/lib** folder to your web library folder **WebContent/WEB-INF/lib**. You will also need to add Javax transaction API, Java Mail API, JDBC drivers, wfengineserver.jar, jvx.jar and appsclient.jar to your web library folder.

After your dependencies are configured, create a new Java package in the src folder of the project. Set **com.sibvisions.quartz** as name. Create a new Java class in this package. The name of the class should be **WorkflowJob**.

Use following source code:

```
public class WorkflowJob implements Job
{
    public void execute(JobExecutionContext context) throws
JobExecutionException
    {
        JobDataMap jobDataMap = context.getJobDetail().getJobDataMap();

        try
        {
            // start workflow
```

```
WorkflowEngine.getWorkflowEngine("QuartzIntegration").startWorkflow("quartz"
,
BigDecimal.valueOf(jobDataMap.getInt("workflowId")),
new Object[] { jobDataMap.getString("mailTo") });
        }
        catch (Exception ex)
        {
            LoggerFactory.getLogger(WorkflowJob.class).error(ex);
        }
    }
}
```

The WorkflowJob is a simple Java class, that instantiates WorkflowEngine for a specific application (**QuartzIntegration**) and starts workflow for a specific user (**quartz**) with configured workflow ID and email parameter. The configuration of parameters will be explained later.

To complete our example, we need to start the Quartz Scheduler. To do this, add following to your Deploymentdescriptor in **WebContent/WEB-INF/web.xml**:

```
<listener>
    <listener-
class>org.quartz.ee.servlet.QuartzInitializerListener</listener-class>
  </listener>
```

Configuration

Job Configuration

Now it's time to configure the Quartz job. To do this, create the folder *classes* in **WebContent/WEB-INF** if missing. In **WebContent/WEB-INF/classes** create *quartz.properties* file with following content:

```
org.quartz.threadPool.threadCount=5

org.quartz.plugin.jobInitializer.class=org.quartz.plugins.xml.XMLSchedulingD
ataProcessorPlugin
org.quartz.plugin.jobInitializer.fileNames=quartz_data.xml
org.quartz.plugin.jobInitializer.wrapInUserTransaction=false
```

In **WebContent/WEB-INF/classes** create *quartz_data.xml* with following content:

```
<?xml version="1.0" encoding="UTF-8"?>

<job-scheduling-data
    xmlns="http://www.quartz-scheduler.org/xml/JobSchedulingData"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.quartz-scheduler.org/xml/JobSchedulingData
```

```
http://www.quartz-scheduler.org/xml/job_scheduling_data_2_0.xsd"
    version="1.8">

    <schedule>
        <job>
            <name>WorkflowJob</name>
            <job-class>com.sibvisions.quartz.WorkflowJob</job-class>
            <job-data-map>
               <entry>
                   <key>mailTo</key>
                   <value>jozef.dorko@sibvisions.com</value>
               </entry>
               <entry>
                   <key>workflowId</key>
                   <value>133</value>
               </entry>
           </job-data-map>
        </job>
        <trigger>
            <cron>
                <name>WorkflowJob</name>
                <job-name>WorkflowJob</job-name>
                <cron-expression>0/10 * * * * ?</cron-expression>
            </cron>
        </trigger>
    </schedule>

</job-scheduling-data>
```

As you can see, we pass two parameters (*mailTo*, *workflowId*) to the job class. Quartz will use configured trigger and starts our job every ten seconds.

Now, the job configuration is done, and Quartz will start our job. The last step is datasource configuration for your workflow engine.

Datasource Configuration

The datasource is needed to run a workflow. It will be configured via *config.xml*. To get everything in place, create the folder structure:

```
WebContent
  |- WEB-INF
      |- rad
          |- apps
              |- QuartzIntegration
```

This is a standard JVx application structure. It is very important that the folder *QuartzIntegration* exists because this is the application name used in our *WorkflowJob* class:

```
WorkflowEngine.getWorkflowEngine("QuartzIntegration")
```

Put the config.xml in folder **WebContent/WEB-INF/rad/apps/QuartzIntegration** The file contains the application configuration, e.g:

```
<?xml version="1.0" encoding="UTF-8"?>

<application>
  <securitymanager>
    <class>com.sibvisions.rad.server.security.DBSecurityManager</class>
<accesscontroller>com.sibvisions.apps.server.object.DBWorkScreenAccess</acce
sscontroller>
    <passwordalgorithm>SHA</passwordalgorithm>
    <database datasource="oracle_project"/>
  </securitymanager>
  <datasource>
    <db name="oracle">
      <url>jdbc:oracle:thin:@localhost:1521:xe</url>
      <username>appl</username>
      <password>appl</password>
    </db>
    <db name="oracle_project">
      <url>jdbc:oracle:thin:@192.168.1.35:1521:xe</url>
      <username>job</username>
      <password>job</password>
    </db>
  </datasource>
  <mail>
    <smtp>
      <host>mail.server.com</host>
      <port>587</port>
      <username></username>
      <password></password>
      <tlsenabled>true</tlsenabled>
      <defaultsender>Quartz Job &lt;noreply@server.com&gt;</defaultsender>
    </smtp>
  </mail>
</application>
```

Logging

If you want to see some details about job execution, you should configure logging. Simply create a file with the name **log4j.xml** in the *src* folder of your Eclipse project. Put following content into it:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
```

```
  <appender name="default" class="org.apache.log4j.ConsoleAppender">
    <param name="target" value="System.out"/>
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="[%p] %d{dd MMM hh:mm:ss.SSS aa}
%t [%c]%n%m%n%n"/>
    </layout>
  </appender>

 <logger name="org.quartz">
   <level value="info" />
 </logger>

  <root>
    <level value="info" />
    <appender-ref ref="default" />
  </root>


</log4j:configuration>
```

Now you're ready.

Testing

You have two options to test the Quartz job. First, use Eclipse to export the project as **war** file. This file can be deployed manually to any Java application server.

The second option is to run an embedded application server directly in Eclipse. To do this, configure an application server and add your project to this server. This makes it possible to debug your code and test if everything works as expected.

From:
https://doc.sibvisions.com/ - **Documentation**

Permanent link:
https://doc.sibvisions.com/workflow/quartz

Last update: **2020/06/15 10:24**