How to… run a Job with JQM

This document describe the installation of the JQM package and how to using it in a Maven project with a local node.

Table of content

[2 Prepare your project 3](#_Toc373511170)

[3 Prepare your job 4](#_Toc373511171)

[4 Store your job 6](#_Toc373511172)

[5 JQM, go ! 7](#_Toc373511173)

[5.1 Create your job definition 7](#_Toc373511174)

[5.1.1 The xml file 7](#_Toc373511175)

[5.1.2 JQM command line 8](#_Toc373511176)

[5.2 Enqueue your principal job 8](#_Toc373511177)

[5.3 You will see the Nirvana 8](#_Toc373511178)

[6 Recurrent problems 9](#_Toc373511179)

[7 Bug report 9](#_Toc373511180)

# Description of the JQM package

In the JQM directory three directories are present. In the root directory you can see the “jqm.jar” file that will be used to run the JQM command lines.

## conf directory

Contains a “db.properties” file where JQM connection information must be set. JQM can’t run successfully if this information are not set correctly.

Fields:

* javax.persistence.driver=database driver named “jdbcdriver.jar” in the lib directory
* javax.persistence.url=url to connect JQM at the database
* javax.persistence.user=database user
* javax.persistence.password=user password

## lib directory

Contains all the JQM dependencies, the xsd file named “res.xsd” and the database driver named “jdbcdriver.jar”. This is an internal directory, nothing must be added.

## jobs directory

Must contains all the jobs, preferably, one directory by job containing the job jar file, and potentially lib directory or a pom.xml.

# Prepare your project

JQM is a simple batch manager easily usable in your maven project.

To easily include JQM in your dependencies, add this dependency:

<dependency>

<groupId>com.enioka.jqm</groupId>

<artifactId>jqm-api</artifactId>

<version>1.1.2</version>

</dependency>

This dependency is THE necessary dependency to use JQM.

This dependency includes:

* getDefaultConnection: To be connected relatively to the jndi insert in the database.
* addDeliverable: To add a deliverable
* sendMsg: To insert a message in the database corresponding to the current job
* sendProgress: Same thing but to insert the job progression
* enQueue: To enqueue a job

If you want to use all the JQM possibilities, you can add another dependency:

<dependency>

<groupId>com.enioka.jqm</groupId>

<artifactId>jqm-clientapi</artifactId>

<version>1.1.2</version>

</dependency>

This dependency includes some useful methods to:

* Change the job priority
* Another way to enqueue a job
* Change the job position
* Get deliverables generated by your job
* …

For more information, the complete documentation is available on Confluence or GitHub.

# Prepare your job

Some rules:

* Your main class must extends the JobBase API.
* Your “main” method must be called “start” and must be overridden.

Example:

* Parent job

import java.util.HashMap;

import java.util.Map;

import com.enioka.jqm.api.JobBase;

public class Caller extends JobBase

{

@Override

public void start()

{

Map<String, String> params = new HashMap<String, String>();

params.put(“myParam1”, “parameter1”);

params.put(“myParam2”, “parameter2”);

// enQueue is a JobBase method.

// It is used to enqueue a Job, here “Called”.

enQueue(“Called”, “Steve”, null, null, null, null, null,

null, null, null, null, params);

System.out.println(“Caller: Enqueue finish. Called job will

start!”);

System.out.println(“Caller: my parameters are ” +

params.get(“myParam1”) + “ & ” +

params.get(“myParam2”) + “!”);

}

}

* Child job

import com.enioka.jqm.api.JobBase;

public class Called extends JobBase

{

@Override

public void start()

{

System.out.println(“Called: Called job launched by Steve!”);

}

}

Here, the Caller job, will enqueue the Called job. You will see appear in the logs:

* “Caller: Enqueue finish. Called job will start!”
* “Caller: my parameters are parameter1 & parameter2!”
* “Called: Called job launched by Steve!”

In this example you can see:

* A map named “params”: In this map, you can enter all the parameters that you need to run your project.
* The enQueue method: This function is a method of the JobBase API. The first and the second parameters are mandatory to use this method. The first parameter is relative to the job name that you want to enqueue. The second parameter corresponds to the user that wants to enqueue the job.

# Store your job

In the JQM package you can see a jobs directory. In this directory you must a directory for each jobs.

Some rules:

* Your jar must contains a lib directory OR a pom.xml (cf below) file.
* Your job directory must contains an xml file to describe your job.

Example:

jobs/

jobs/callerJob/

jobs/callerJob/caller\_job.jar

jobs/callerJob/caller\_job.xml

unix-term-$ jar tf ./caller\_job.jar

META-INF/

META-INF/MANIFEST.MF

META-INF/…

lib/

lib/myjar.jar

lib/mysecondjar.jar

lib/…

Caller.class

Or

META-INF/

META-INF/MANIFEST.MF

META-INF/maven/

META-INF/maven/mygroupid/

META-INF/maven/mygroupid/myartifactid/

META-INF/maven/mygroupid/myartifactid/pom.xml

META-INF/maven/mygroupid/myartifactid/pom.properties

Caller.class

# JQM, go !

## Create your job definition

### The xml file

This xml file reference all the information about your job. Some information are mandatory as:

* Path: Relative path to your jar.
* Name: Name of the job.
* CanBeRestarted: Boolean to define if the job can be restarted.
* JavaClassName: Name of the entry point class.
* FilePath: file path to your job directory.
* Highlander: Boolean to define if only one job can be enqueued in the same queue.

Careful: all the xml fields are mandatory. The data can be null but the fields must appear.

Here:

<jqm>

<jar>

<path>callerJob/caller\_job.jar</path>

<filePath>callerJob/</filePath>

<jobdefinitions>

<jobDefinition>

<name>Caller</name>

<description>Test to the how to…</description>

<canBeRestarted>true</canBeRestarted>

<javaClassName>Caller</javaClassName>

<maxTimeRunning>42</maxTimeRunning>

<application>ApplicationTest</application>

<module>TestModule</module>

<other1>Jqm1</other1>

<other2>Jqm2</other2>

<other3>Jqm3</other3>

<highlander>false</highlander>

<parameters>

<parameter>

<key>myParam1</key>

<value>Donald</value>

</parameter>

<parameter>

<key>myParam2</key>

<value>Goofy</value>

</parameter>

</parameters>

</jobDefinition>

</jobdefinitions>

</jar>

</jqm>

### JQM command line

This command line will insert a job definition (JobDef table in the database) with all the job information.

Command line:

java -jar jqm.jar *myNode* –xml *filepath/to/myjob.xml*

Here:

java -jar jqm.jar testNode–xml ./jobs/callerProject/caller\_job.xml

## Enqueue your principal job

This command line will insert in the database a job instance (JobInstance in the database). The job in enqueued and should be catch by a running node to be performed.

Command line :

java -jar jqm.jar *myNode* –enqueue *JobName*

Here :

java -jar jqm.jar testNode–enqueue Caller

## You will see the Nirvana

This command line will run your node. The node will poll the queues and run the “SUBMITTED” jobs.

Command line:

java -jar jqm.jar *myNode*

Here :

java -jar jqm.jar testNode

# Recurrent problems

* When starting JQM, if “address already in use” error appear. Change the ports of your nodes (by default a node is on the port 1789)
* When starting JQM, if “Unable to build EntityManager factory” error appear, the connection with the database is impossible. Check the information in the conf/db.properties file.
* Problem with the password during the execution of the SUBMITTED job: In the DatabaseProp table, the field pwd must be changed.
* Problem with the download of the dependencies during the execution: Your nexus or depositories configuration must be wrong. Check your pom.xml or your setting.xml.

# Bug report

Marc-Antoine Gouillart: [marc-antoine.gouillart@enioka.com](mailto:marc-antoine.gouillart@enioka.com)