Getting Started with Quartz Scheduler

Version 2.2.1

This document applies to Quartz Scheduler Version 2.2.1 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: QS-GS-221-20141015

Table of Contents

About Quartz Scheduler.....5
What is Quartz Scheduler?.....6
Quartz Features.....6

Installing and Using Quartz Scheduler.....9
Downloading and Installing Quartz.....10
Configuring Quartz Scheduler.....11
Starting a Sample Application.....11

1 About Quartz Scheduler

- What is Quartz Scheduler? 6
- Quartz Features 6

What is Quartz Scheduler?

Quartz Scheduler is a richly featured, open source job scheduling library that can be integrated within virtually any Java application - from the smallest stand-alone application to the largest e-commerce system. Quartz can be used to create simple or complex schedules for executing tens, hundreds, or even tens-of-thousands of jobs; jobs whose tasks are defined as standard Java components that may execute virtually anything you may program them to do. The Quartz Scheduler includes many enterprise-class features, such as support for JTA transactions and clustering.

What Can Quartz Do For You?

If your application has tasks that need to occur at given moments in time, or if your system has recurring maintenance jobs then Quartz may be your ideal solution.

Sample uses of job scheduling with Quartz:

- Driving Process Workflow: As a new order is initially placed, schedule a Job to fire in exactly 2 hours, that will check the status of that order, and trigger a warning notification if an order confirmation message has not yet been received for the order, as well as changing the order's status to 'awaiting intervention'.
- System Maintenance: Schedule a job to dump the contents of a database into an XML file every business day (all weekdays except holidays) at 11:30 PM.
- Providing reminder services within an application.

Quartz Features

Runtime Environments

- Quartz can run embedded within another free standing application
- Quartz can be instantiated within an application server (or servlet container), and participate in XA transactions
- Quartz can run as a stand-alone program (within its own Java Virtual Machine), to be used via RMI
- Quartz can be instantiated as a cluster of stand-alone programs (with load-balance and fail-over capabilities) for the execution of jobs

Job Scheduling

Jobs are scheduled to run when a given trigger occurs. Triggers can be created with nearly any combination of the following directives:

- At a certain time of day (to the millisecond)
- On certain days of the week

- On certain days of the month
- On certain days of the year
- Not on certain days listed within a registered Calendar (such as business holidays)
- Repeated a specific number of times
- Repeated until a specific time/date
- Repeated indefinitely
- Repeated with a delay interval

Jobs are given names by their creator and can also be organized into named groups. Triggers may also be given names and placed into groups, in order to easily organize them within the scheduler. Jobs can be added to the scheduler once, but registered with multiple triggers. Within an enterprise Java environment, jobs can perform their work as part of a distributed (XA) transaction.

Job Execution

- Jobs can be any Java class that implements the simple Job interface, leaving infinite possibilities for the work your jobs can perform.
- Job class instances can be instantiated by Quartz, or by your application's framework.
- When a trigger occurs, the scheduler notifies zero or more Java objects implementing the JobListener and TriggerListener interfaces (listeners can be simple Java objects, or EJBs, or JMS publishers, etc.). These listeners are also notified after the job has executed.
- As jobs are completed, they return a JobCompletionCode which informs the scheduler of success or failure. The JobCompletionCode can also instruct the scheduler of any actions it should take based on the success/fail code - such as immediate re-execution of the job.

Job Persistence

- The design of Quartz includes a JobStore interface that can be implemented to provide various mechanisms for the storage of jobs.
- With the use of the included JDBCJobStore, all jobs and triggers configured as "non-volatile" are stored in a relational database via JDBC.
- With the use of the included RAMJobStore, all jobs and triggers are stored in RAM and therefore do not persist between program executions - but this has the advantage of not requiring an external database.

Transactions

- Quartz can participate in JTA transactions, via the use of JobStoreCMT (a subclass of JDBCJobStore).

- Quartz can manage JTA transactions (begin and commit them) around the execution of a job, so that the work performed by the Job automatically happens within a JTA transaction.

Clustering Features

- Provides fail-over.
- Provides load balancing.
- Quartz's built-in clustering features rely upon database persistence via JDBCJobStore (described above).
- Terracotta extensions to Quartz provide clustering capabilities without the need for a back-end database.

Listeners & Plug-Ins

- Applications can catch scheduling events to monitor or control job/trigger behavior by implementing one or more listener interfaces.
- The Plug-In mechanism can be used add functionality to Quartz, such keeping a history of job executions, or loading job and trigger definitions from a file.
- Quartz ships with a number of "factory built" plug-ins and listeners.

2 Installing and Using Quartz Scheduler

- Downloading and Installing Quartz 10
- Configuring Quartz Scheduler 11
- Starting a Sample Application 11

Downloading and Installing Quartz

First, download the most recent stable release from http://quartz-scheduler.org/downloads. Registration is not required. Unpack the distribution and install it so that your application can see it.

The Quartz JAR Files

The Quartz package includes a number of jar files, located in root directory of the distribution. The main Quartz library is named quartz-xxx.jar (where xxx is a version number). In order to use any of Quartz's features, this jar must be located on your application's classpath.

If you use Quartz primarily within an application server environment, you can place the Quartz JAR within your application (.ear or .war file). If you want to make Quartz available to many applications, make sure it is in the classpath of your application server. If you are using it with a stand-alone application, place it in the application's classpath along with all of the other JARs your application depend on.

Quartz depends on a number of third-party libraries (in the form of jars) which are included in the distribution .zip file in the 'lib' directory. To use all the features of Quartz, these jars must also exist on your classpath. If you're building a stand-alone Quartz application, add all of the libraries to the classpath.

Note: In an application server environment, unexpected results might occur if the environment includes different versions of the same jar. For example, WebLogic includes an implementation of J2EE (inside weblogic.jar) which might differ from the one in servlet.jar. In this case, it is usually better to leave servlet.jar out of your application, so you know which classes are being utilized.

The Properties File

Quartz uses a properties file called quartz.properties. This file is not necessary at first, but it is required when you want to use anything other than the basic configuration. When you use this file, it must be in your classpath.

If you're building a web application (i.e., in the form of a .war file), you can put the quartz.properties file in the WEB-INF/classes folder to place it in the application's classpath.

Tip: If you develop your application using WebLogic Workshop, you can keep your configuration files (including quartz.properties) in a project under the root of your application. When you package your application into an .ear file, the configuration project will be packaged into a .jar that is included in the final .ear. Doing this will automatically put the quartz.properties file in your classpath.

Configuring Quartz Scheduler

To configure Quartz, you edit the quartz.properties file and place it in your application's classpath. For information about adding the properties file to your classpath, see "Downloading and Installing Quartz" on page 10.

The Quartz Scheduler distribution package includes several example quartz.properties files in the examples/ directory.

The following example shows the contents of a basic quartz.properties file:

```
org.quartz.scheduler.instanceName = MyScheduler
org.quartz.threadPool.threadCount = 3
org.quartz.jobStore.class = org.quartz.simpl.RAMJobStore
```

This example specifies the following properties:

- The `org.quartz.scheduler.instanceName` property assigns the name "MyScheduler" to the scheduler.
- The `org.quartz.threadPool.threadCount` property allocates three threads in the thread pool, meaning that a maximum of three jobs can be run simultaneously.
- The `org.quartz.jobStore.class` property configures the scheduler to use the RAMJobStore for data storage. This means that the scheduler will keep its job and trigger data in memory rather than in a database. For more information about job stores, see “Working with Job Stores” in the *Quartz Scheduler Developer Guide*.

For descriptions of all the properties that a quartz.properties file can include, see the *Quartz Scheduler Configuration Guide*.

Starting a Sample Application

The following code obtains an instance of the Scheduler, starts it, then shuts it down.

QuartzTest.java

```
import org.quartz.Scheduler;
import org.quartz.SchedulerException;
import org.quartz.impl.StdSchedulerFactory;
import static org.quartz.JobBuilder.*;
import static org.quartz.TriggerBuilder.*;
import static org.quartz.SimpleScheduleBuilder.*;
```

Note: Once you obtain a scheduler using StdSchedulerFactory.getDefaultScheduler(), your application will not terminate until you call scheduler.shutdown(). This is because there will be active threads.

Note the use of static imports. These come into play in the next code sample.

If you have not set up logging, log information is sent to the console. Your output will look similar to this:

```
[INFO] 21 Jan 08:46:27.857 AM main [org.quartz.core.QuartzScheduler]
Quartz Scheduler v.2.0.0-SNAPSHOT created.
[INFO] 21 Jan 08:46:27.859 AM main [org.quartz.simpl.RAMJobStore]
RAMJobStore initialized.
[INFO] 21 Jan 08:46:27.865 AM main [org.quartz.core.QuartzScheduler]
Scheduler meta-data: Quartz Scheduler (v2.0.0) 'Scheduler' with instanceId 'NON_CLUSTERED'
  Scheduler class: 'org.quartz.core.QuartzScheduler' - running locally.
  NOT STARTED.
  Currently in standby mode.
  Number of jobs executed: 0
  Using thread pool 'org.quartz.simpl.SimpleThreadPool' - with 50 threads.
  Using job-store 'org.quartz.simpl.RAMJobStore' - which does not support
        persistence. and is not clustered.
[INFO] 21 Jan 08:46:27.865 AM main [org.quartz.impl.StdSchedulerFactory]
Quartz scheduler 'Scheduler' initialized from default resource file in Quartz
        package: 'quartz.properties'
[INFO] 21 Jan 08:46:27.866 AM main [org.quartz.impl.StdSchedulerFactory]
Quartz scheduler version: 2.0.0
[INFO] 21 Jan 08:46:27.866 AM main [org.quartz.core.QuartzScheduler]
Scheduler Scheduler_$_NON_CLUSTERED started.
[INFO] 21 Jan 08:46:27.866 AM main [org.quartz.core.QuartzScheduler]
Scheduler Scheduler_$_NON_CLUSTERED shutting down.
[INFO] 21 Jan 08:46:27.866 AM main [org.quartz.core.QuartzScheduler]
Scheduler Scheduler_$_NON_CLUSTERED paused.
[INFO] 21 Jan 08:46:27.867 AM main [org.quartz.core.QuartzScheduler]
Scheduler Scheduler_$_NON_CLUSTERED shutdown complete.
```

To do something, you need to provide code between the `start()` and `shutdown()` calls. You will also need to allow some time for the job to be triggered and executed before calling shutdown(). For a simple example like this, you might call `Thread.sleep(60000)`.

```
// define the job and tie it to our HelloJob class
JobDetail job = newJob(HelloJob.class)
    .withIdentity("job1", "group1")
    .build();
// Trigger the job to run now, and then repeat every 40 seconds
Trigger trigger = newTrigger()
    .withIdentity("trigger1", "group1")
    .startNow()
    .withSchedule(simpleSchedule()
            .withIntervalInSeconds(40)
            .repeatForever())
    .build();
// Tell quartz to schedule the job using our trigger
scheduler.scheduleJob(job, trigger);
```

For additional code samples, see *Quartz Scheduler Example Programs and Code Samples*.