ACube

# Normes de développement Quartz pour LISE v3.x

**Version 1.4 du 22/02/2010**

**Etat : Validé**

## SUIVI DES MODIFICATIONS

| Version | Rédaction | Description | Vérification | Date |
|---|---|---|---|---|
| 1.0 | T.Rigal | Initialisation | | 14/03/07 |
| 1.1 | G.Pasquereau | Compléments et précisions | | 19/03/07 |
| 1.2 | M.Fraudeau | Quartz Lise3 | | 06/08/08 |
| 1.3 | JP.Wilsch | Compléments Lise 3 + Supervisor | | 08/01/09 |
| 1.3.1 | G.Pasquereau | Réorganisation | | 26/02/09 |
| 1.4 | G.Pasquereau | Actualisation | | 22/01/10 |

## LISTE DE DIFFUSION

| Organisation | Nom | Info | Commentaire | Validation |
|---|---|---|---|---|
| | | ☐ | ☐ | ☐ |
| | | ☐ | ☐ | ☐ |
| | | ☐ | ☐ | ☐ |

# SOMMAIRE



# DOCUMENTS DE REFERENCE

| Version | Titre |
|---|---|
| | |
| | |
| | |
| | |

# 1 INTRODUCTION

Ce document explique comment planifier des tâches dans une application ACube en s'appuyant sur la librairie Quartz.

Quartz est un ordonnanceur de traitement qui s'inscrit dans le projet Open source « OpenSymphony ».

Après une brève présentation du composant, le document s'attachera aux problématique de mise en œuvre et de supervision des traitement ainsi qu'aux modalités de production d'un livrable applicatif utilisant Quartz.

# 2 PRESENTATION DE QUARTZ

Il existe trois objets principaux pour Quartz. Le scheduler qui correspond à l'ordonnanceur des tâches, les jobs qui correspondent aux tâches à exécuter et les triggers qui déclenchent l'exécution d'une tâche.

## 2.1 L'OBJET SCHEDULER

Le *scheduler* correspond donc à l'ordonnanceur de tâches. Il est indispensable à la planification, c'est sur lui que sont fixés les jobs ainsi que les triggers. Pour nos exemples, nous utiliserons le *scheduler* par défaut qui ne gère pas la persistance des tâches.

## 2.2 L'OBJET JOB

Il faut maintenant créer une tâche qui sera planifiée ultérieurement. Le GabaritProjetJ2EE contient à titre d'exemple une classe de Job : *acube.projet.quartz.job.SampleJob*. Cette classe implémente l'interface Job, qui lui confère la méthode « *execute* » à implémenter. Ici un simple "bonjourN." s'affiche dans une console (où N est compteur s'incrémentant à chaque exécution du Job).

## 2.3 L'OBJET JOBDETAIL

Pour instancier le Job, nous avons besoin d'une nouvelle classe appelée JobDetail. Cette classe est indispensable à l'activation du Job. Dans son principe le JobDetail est une sorte de « Factory » pour les objets Job à instancier et à exécuter.

## 2.4 L'OBJET TRIGGER

Il s'agit de créer des déclencheurs des tâches implémentées préalablement. Il est à noter qu'un *job* peut être déclenché par plusieurs *triggers*, mais de façon non réciproque, un *trigger* ne peut pas déclencher plusieurs *Jobs*.

Il faut savoir qu'il existe deux types de triggers :

- *SimpleTrigger* : ce type de *trigger* se base sur des intervalles réguliers et sur un nombre de répétitions précis
- *CronTrigger* : celui-ci offre plus de liberté sur la planification, puisqu'il se base sur les jours du calendrier

**Les *triggers* utilisés dans les applications ACube seront obligatoirement de type *CronTrigger*.**

# 3 DECLARATION D'UN JOB ET DE SES DECLENCHEURS AUPRES DE SPRING

La déclaration des jobs et des triggers permettant le déclenchement des jobs Quartz se fait par l'intermédiaire d'un fichier de configuration appContext.xml dédié au framework Spring, inclus dans le framework Lise V3.x.

La déclaration d'un job se réalise par l'intermédiaire d'un objet de type JobDetail. Les objets JobDetail contiennent toutes les informations nécessaires pour lancer un job.

Spring permet la création d'un JobDetailBean.

## 3.1 CREATION DU SERVICE UTILISÉ PARQUARTZ

Créer un service métier qui sera utilisé par le job Quartz.

Exemple de création d'un service dans le fichier appContext.xml :

```
<bean id="quartzService"
        class="acube.projet.business.service.QuartzServiceImpl">
        <constructor-arg ref="stagiaireALaCandidatureDAO" />
        <constructor-arg ref="rechercherCandidatureDAO" />
        <constructor-arg ref="donneesMailValidationStagiaireDAO" />
        <constructor-arg ref="donneesMailCaduciteOffreDAO" />
        <constructor-arg ref="oMailDAO" />
</bean>
```

## 3.2 CREATION DU JOB QUI UTILISERA LE SERVICE

3 propriétés sont à renseigner lors de la déclaration du bean correspondant au Job dans le fichier appContext.xml :

- JobClass : La classe du Job
- Description : Une description succinte du Job
- JobDataAsMap : Une map permettant de renseigner la valeur des attributs de la classe du Job

Dans le fichier appContext.xml :

```
    <bean id="quartzAction"
class="org.springframework.scheduling.quartz.JobDetailBean">
        <property name="jobClass">
            <value>acube.projet.web.quartz.job.QuartzJob</value>
        </property>
        <property name="description">
            <value>Description du job</value>
        </property>
        <property name="jobDataAsMap">
            <map>
```

```
                <entry key="quartzService">
                    <ref bean="quartzService" />
                </entry>
            </map>
        </property>
    </bean>
```

Dans le projet :

```
package acube.projet.web.quartz.job.QuartzJob;

public class QuartzJob extends QuartzJobBean {

  private QuartzService quartzService;

    /*
     * Setter of the quartzService
     *
     * @param quartzService
     *            The quartzService to set.
     */
    public void setQuartzService(
            QuartzService quartzService) {


  protected void executeInternal(JobExecutionContext ctx)
  throws JobExecutionException {
      // do the actual work
      this.quartzService.doMyBusiness(…);
  }
}
```

## 3.3 CREATION D'UN DECLENCHEUR POUR UN JOB (TRIGGER)

2 propriétés sont à renseigner lors de la déclaration du bean correspondant au Trigger qui déclenchera le Job, dans le fichier appContext.xml :

- JobDetail : La référence vers le JobDetailBean décrivant le Job
- cronExpression : L'expression CRON décrivant la séquence de déclenchement du trigger

Dans le fichier appContext.xml :

```
    <bean id="triggerTache"
class="org.springframework.scheduling.quartz.CronTriggerBean">
        <property name="jobDetail" ref=" quartzJob "/>
        <!-- Exécution tous les jours à 5 secondes -->
        <property name="cronExpression" value="0/5 * * ? * *"/>
    </bean>
```

## 3.4 CREATION D'UNE FABRIQUE D'ORDONNANCEUR (SCHEDULER)

La dernière étape est de déclarer dans le appContext.xml une fabrique d'ordonnanceur, qui permettra à Spring de créer et gérer l'ordonnanceur de l' application, en lui indiquant les triggers déclarés précédemment.

Dans le fichier appContext.xml :

```
        <bean class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
            <property name="triggers">
                <list>
                    <ref bean="triggerTache" />
                </list>
            </property>
        </bean>
```

## 3.5 LE FICHIER QUARTZ.PROPERTIES

Ce fichier contient un ensemble de propriétés pré-affectées selon des valeurs compatibles avec une meilleure intégration de Quartz aux environnements de production des applications ACube. Sans détailler ici l'ensemble des valeurs choisies, il faut l'importance de :

- la gestion des logs (propriétés « `org.quartz.plugin.jobHistory.jobSuccessMessage` » et « `org.quartz.plugin.jobHistory.jobFailedMessage` »)

Voici le fichier *quartz.properties* actuellement utilisé :

```
#============================================================================
# Configuration Main Scheduler Properties
#============================================================================
org.quartz.scheduler.instanceName = MyScheduler
org.quartz.scheduler.instanceId = AUTO
#============================================================================
# Configuration ThreadPool
#============================================================================
org.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool
org.quartz.threadPool.threadCount = 1
org.quartz.threadPool.threadPriority = 5
#============================================================================
# Configuration JobStore
#============================================================================
org.quartz.jobStore.misfireThreshold = 60000
org.quartz.jobStore.class = org.quartz.simpl.RAMJobStore
#============================================================================
# Configuration Plugins
#============================================================================
# configuration des logs
org.quartz.plugin.jobHistory.class =
org.quartz.plugins.history.LoggingJobHistoryPlugin
org.quartz.plugin.jobHistory.jobSuccessMessage = Traitement {1}.{0} du {2, date,
dd/MM/yyyy HH:mm:ss} code_retour=OK
org.quartz.plugin.jobHistory.jobFailedMessage = Traitement {1}.{0} du {2, date,
```

```
dd/MM/yyyy HH:mm:ss} code_retour=ERREUR
```

Remarques :

- Dans la majorité des projets, seule la propriété "org.quartz.scheduler.instanceName" est à modifier.
- La structure des messages de logs doit être conservée telle que définie.
- L'emplacement de ce fichier à l'exécution est à la racine du répertoire contenant les classes Java compilées (WEB-INF/classes), c'est-à-dire au même niveau que les autres fichiers de propriétés utilisées par l'application.

## 3.6 AJOUT DE LA SUPERVISION DES TRAITEMENTS

L'utilisation d'une librairie de supervision des Jobs Quartz permet aux équipes d'exploitation de disposer d'un outil de supervision des jobs Quartz des applications ACube.

Il faut donc ajouter la librairie :

- AcubeQuartzSupervisor 1.0

Cette librairie contient un fichier appContext-supervisor.xml, qui permet de réaliser, par l'intermédiaire de Spring, l'initialisation des classes de supervision. Ce fichier appContext-supervisor.xml sera à déclarer dans le web.xml du projet, comme indiqué au paragraphe suivant.

Ci-dessous le contenu du fichier appContext-supervisor.xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:tx="http://www.springframework.org/schema/tx"
	xsi:schemaLocation="
		http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
		http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-2.5.xsd
		http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">
	<bean id="propertyConfigurer"

	class="org.springframework.web.context.support.ServletContextPropertyPlacehol
derConfigurer" />

	<!--  Quartz Supervisor -->
	<bean id="quartzSupervisor" class="supervisor.impl.QuartzSupervisorImpl">
	</bean>

	<!--  Job Listener -->
	<bean id="quartzJobListener" class="listener.QuartzJobListener">
		<constructor-arg>
```

```
                        <ref bean="quartzSupervisor" />
                </constructor-arg>
        </bean>

        <!--  Publication du supervisor par RMI, sur le port par défaut -->
        <bean id="registry"
class="org.springframework.remoting.rmi.RmiRegistryFactoryBean">
        </bean>
        <bean class="org.springframework.remoting.rmi.RmiServiceExporter">
                <property name="serviceName" value="${supervisor.name}" />
                <property name="service" ref="quartzSupervisor" />
                <property name="serviceInterface" value="supervisor.QuartzSupervisor"
/>
                <property name="registry" ref="registry" />
        </bean>
        <!--  Publication du supervisor par JMX, en utilisant le serveur du conteneur
-->
        <bean id="exporter" class="org.springframework.jmx.export.MBeanExporter">
                <property name="beans">
                        <map>
                                <entry key="supervisors:name=${supervisor.name}"
value-ref="quartzSupervisor" />
                        </map>
                </property>
                <property name="assembler">
                        <bean

        class="org.springframework.jmx.export.assembler.InterfaceBasedMBeanInfoAssemb
ler">
                                <property name="managedInterfaces">
                                        <value>supervisor.QuartzSupervisor</value>
                                </property>
                        </bean>
                </property>
        </bean>
</beans>
```

## 3.7 FICHIER WEB.XML

Le fichier web.xml d'une application Quartz devra référencer les différents fichier de configuration Spring de l'application, notamment le fichier appContext.xml déclarant les jobs et triggers, et le fichier appContext-supervisor.xml, permettant de déclarer les éléments du superviseur d'application Quartz.

Voici une configuration pour lancer Spring :

```
<web-app>
        <display-name>GabaritProjetJ2EEQuartz</display-name>
        <description>GabaritProjetJ2EEQuartz</description>
```

```
        <!--  Indique au ContextLoaderListener (cf plus bas) où sont les fichiers de
configuration Spring -->
        <context-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath*:appContext.xml classpath*:appContext-security.xml
classpath*:appContext-dao.xml classpath*:appContext-supervisor.xml </param-value>
        </context-param>
        <!--  Renseigne dans le fichier de configuration appContext-supervisor.xml le
nom donné au superviseur -->
        <context-param>
            <param-name>supervisor.name</param-name>
            <param-value><!—- <NomDuProjet>Supervisor --></param-value>
        </context-param>

        <listener>
           <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
        </listener>

        <session-config>
                <!-- in minutes-->
                <session-timeout>30</session-timeout>
        </session-config>

</web-app>
```

On remarque dans ce fichier un paramètre de contexte nommé `supervisor.name`. Ce paramètre permet à chaque application de renseigner le nom unique que portera le superviseur d'application Quartz de cette application.

La norme pour nommer ce superviseur est :

<NomDuProjet>Supervisor

Par exemple, pour le projet Phileas, le nom a donner au superviseur sera :

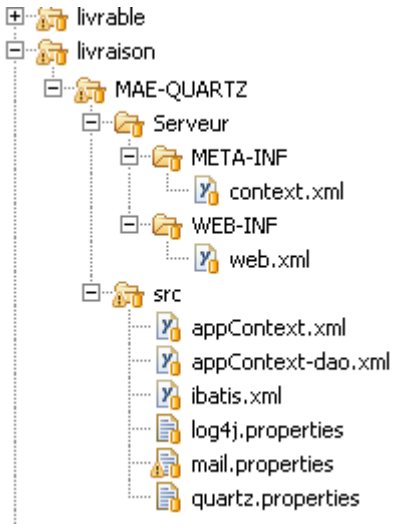
PhileasSupervisor

**IMPORTANT : Ce nom devra être fixé et indiqué dans les spécifications de configuration et de déploiement de l'application.**

## 3.8 TACHES ANT

Voici une proposition à la racine du projet qui permet de regrouper les fichiers taggués pour une utilisation avec Quartz.

-livraison/MAE-QUARTZ contient les fichiers qui viendront remplacer les fichiers de production de l'application.




Avec cette structure, voici la tache ant qui crée le war avec le quartz :

```
        <!-- CREATION de l'archive WAR de l'application  selon la configuration
choisie-->
        <target name="creationWarGenerique" depends="compilation">
                <echo message="************************************" />
                <echo message="CREATION DU FICHIER war" />
                <echo message="************************************" />
                <tstamp>
                        <format property="build.dateLivrable" pattern="yyyyMMdd" />
                </tstamp>

                <mkdir dir="${build.tmp}/ressourceswar" />
                <mkdir dir="./${build.buildRepertoireLivrable}/${configuration}" />
                <copy todir="./${build.buildRepertoireLivrable}/${configuration}"
overwrite="true">
                        <fileset dir="./${build.buildRepertoireLivrable}"
includes="*.xml" />
                        <fileset dir="./${build.buildRepertoireLivrable}"
includes="*.properties" />
                </copy>

                <copy todir="${build.tmp}/ressourceswar" overwrite="true">
                        <fileset dir="Serveur" includes="**/*.xml" />
                </copy>
                <copy todir="${build.tmp}/ressourceswar" overwrite="true">
                        <fileset dir="livraison/${configuration}/Serveur" />
                </copy>
                <copy todir="${build.tmp}/ressourceswar/WEB-INF/classes"
overwrite="true">
                        <fileset dir="src" includes="**/*.xml" />
                        <fileset dir="src" includes="**/*.properties" />
                </copy>
                <copy todir="${build.tmp}/ressourceswar/WEB-INF/classes"
overwrite="true">
```

```
                        <fileset dir="livraison/${configuration}/src" includes="*.*"
/>
                </copy>

                <property name="build.warName" value="livrable-${build.projetNom}-
dynamique-${build.projetVersion}-${build.dateLivrable}.war" />

                <war
destfile="./${build.buildRepertoireLivrable}/${configuration}/${build.warName}"
webxml="${build.tmp}/ressourceswar/WEB-INF/web.xml" duplicate="fail">
                        <lib dir="Serveur/WEB-INF/lib" />
                        <classes dir="${build.tmp}/bin" />
                        <zipfileset dir="${build.tmp}/ressourceswar" />
                        <zipfileset dir="Serveur/templates" prefix="templates" />
                </war>
                <delete dir="${build.tmp}" />
        </target>
```

Lancement de la tache ant qui construit le war pour utiliser quartz.

```
        <!--  taches ANT personnalisee pour le traitement Quartz -->
        <target name="creationWarQuartz" description="CREATION de l'archive WAR de
l'application" depends="compilation">
                <antcall target="creationWarGenerique">
                        <param name="configuration" value="MAE-QUARTZ">
                        </param>
                </antcall>
        </target>
```