Team Learning Session 2 Meeting Notes: **QUARTZ JOBS AND LISTENERS**

TOPICS DISCUSSED:

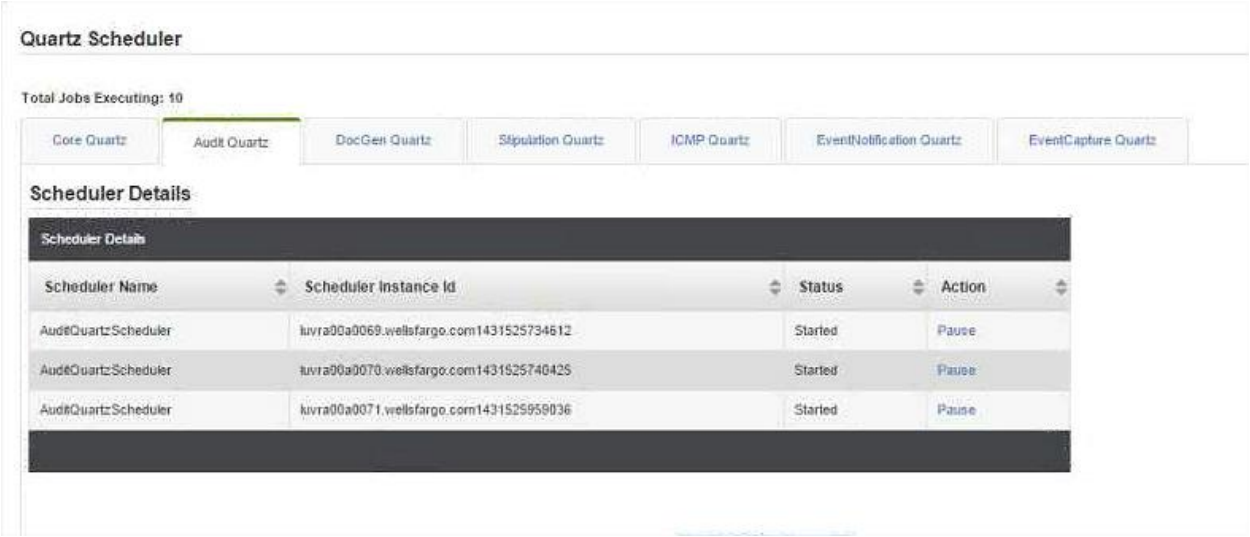
S.NO	Topic
1	Quartz Jobs
2	Quartz Jobs display and Control from Core Tools UI
3	Purpose of the Jobs
4	Schedulers
5	QUARTZ jobs tables
6	Running Jobs locally, New Job, logs, Common Issues
7	CRON Jobs
8	Important Points about Jobs
9	Listeners

Date	**05-13-2015**
Topics Covered by:	Venkata Vanukuri
Document Compiled by:	Krishna Bodduluri
Document Version	1.0

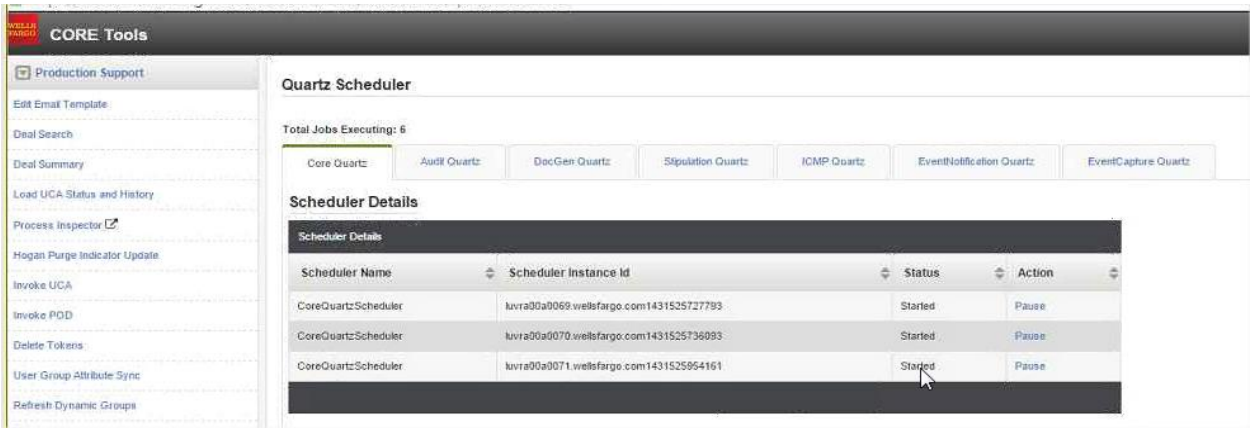
QUARTZ JOBS:

Note: Framework team owns and manages all the Quartz Jobs. We don't own/manage the functionality perspective of jobs that are not related to Admin/Framework Team.

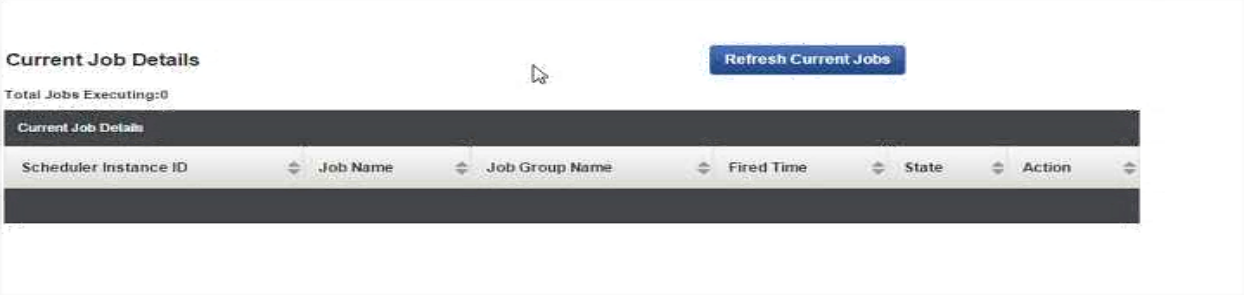
There were 7 total different quartz jobs as below. We can schedule jobs from UI too.



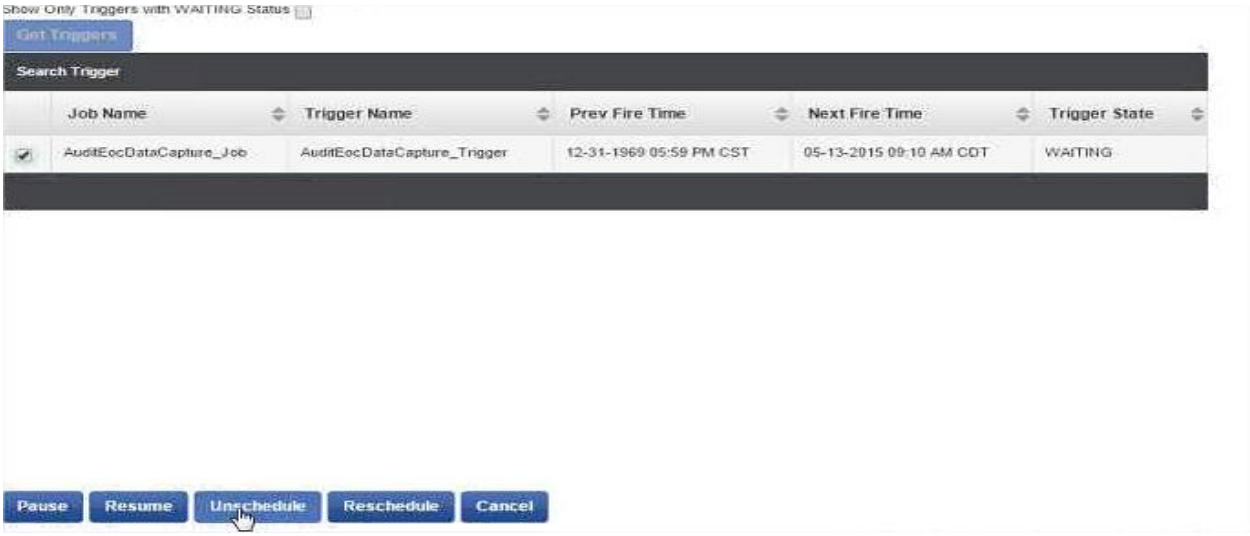
Quartz jobs can be accessed from Core Tools UI.



Quartz Scheduler UI gives more info about Current Job Details. There is also a Refresh jobs option in the UI to get most current job info.



We can Control **(PAUSE, RESUME, UNSCHEDULE, RESCHEDULE, CANCEL)** the jobs from UI.

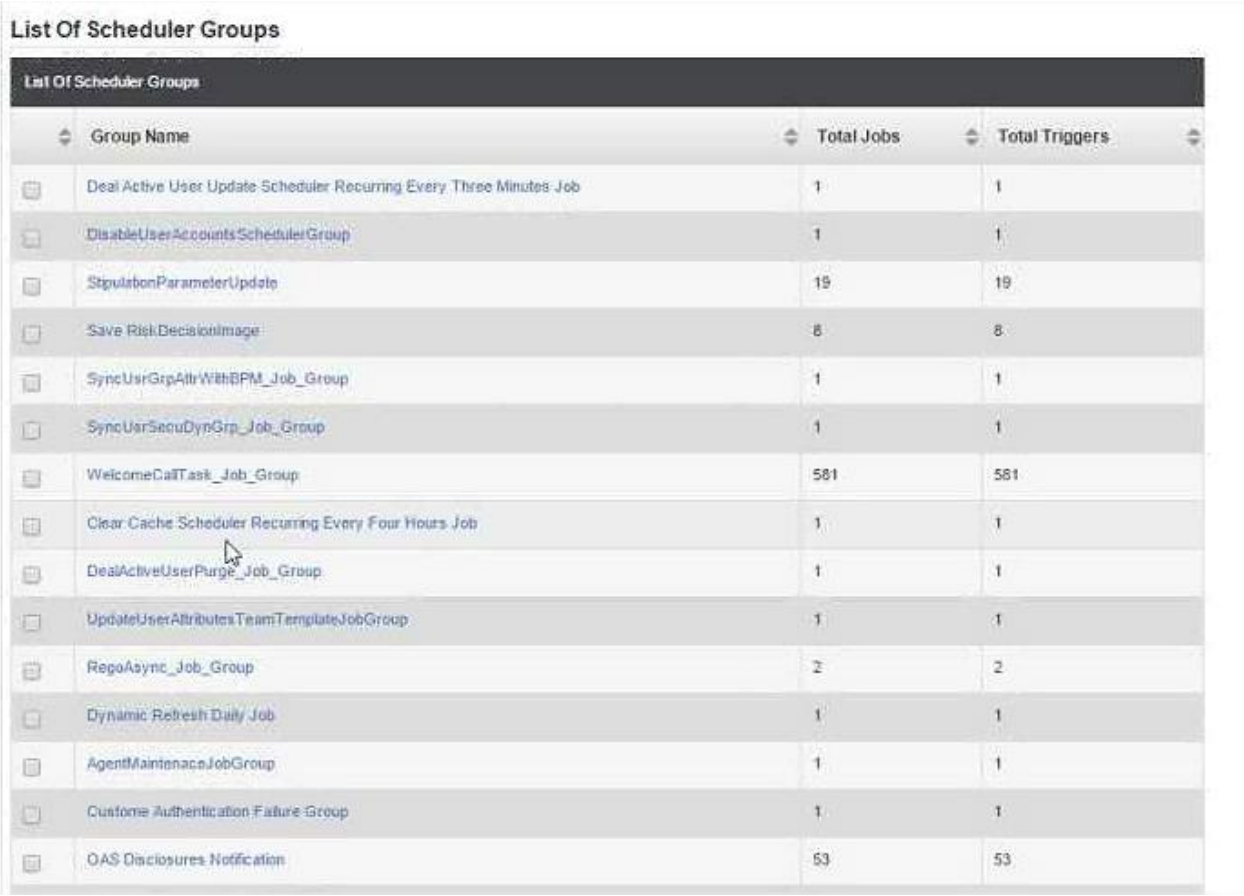


Purpose of having the jobs:

Here are the few reasons to go with Quartz Scheduler Jobs

- Some services may go down sometimes. During the loan process, if the services are down, services are down, several tasks gets created. Even if one service is down, there can be several tasks gets created.
- To bulk complete several system exceptions.
- When service call is failed, System exception gets created for that. Prod support will looks into it. To handle bulk number of System exception tasks.
- To handle sending bulk quantity Emails etc..
- To Clear the Cache.
- Documents upload etc..

In the UI, we have Several Scheduler Groups. Each group can have 1 or more jobs. If we have to develop a new job, pick the right group as placeholder for the new job.



List Of Scheduler Groups

List Of Scheduler Groups

Group Name	Total Jobs	Total Triggers
Deal Active User Update Scheduler Recurring Every Three Minutes Job	1	1
DisableUserAccountsSchedulerGroup	1	1
StipulationParameterUpdate	19	19
Save RiskDecisionImage	8	8
SyncUsrGrpAttrWithBPM_Job_Group	1	1
SyncUsrSecuDynGrp_Job_Group	1	1
WelcomeCallTask_Job_Group	581	581
Clear Cache Scheduler Recurring Every Four Hours Job	1	1
DealActiveUserPurge_Job_Group	1	1
UpdateUserAttributesTeamTemplateJobGroup	1	1
RegoAsync_Job_Group	2	2
Dynamic Refresh Daily Job	1	1
AgentMaintenaceJobGroup	1	1
Custome Authentication Failure Group	1	1
OAS Disclosures Notification	53	53

The “Scheduler Details” tab gives more info about Scheduler name and Instance where the job is running.

Scheduler Details

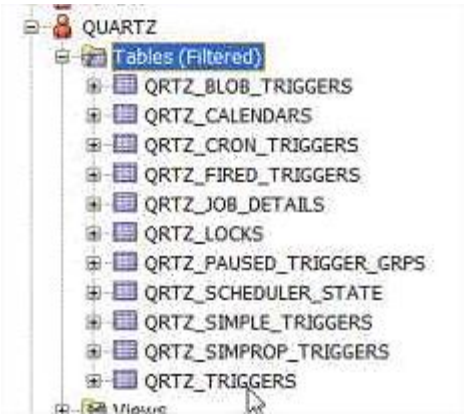
Scheduler Details

Scheduler Name	Scheduler Instance Id	Status	Action
EventCaptureQuartzScheduler	luvra00a0069.wellsfargo.com1431525730016	Stopped	Resume
EventCaptureQuartzScheduler	luvra00a0070.wellsfargo.com1431525738023	Stopped	Resume
EventCaptureQuartzScheduler	luvra00a0071.wellsfargo.com1431525956383	Stopped	Resume

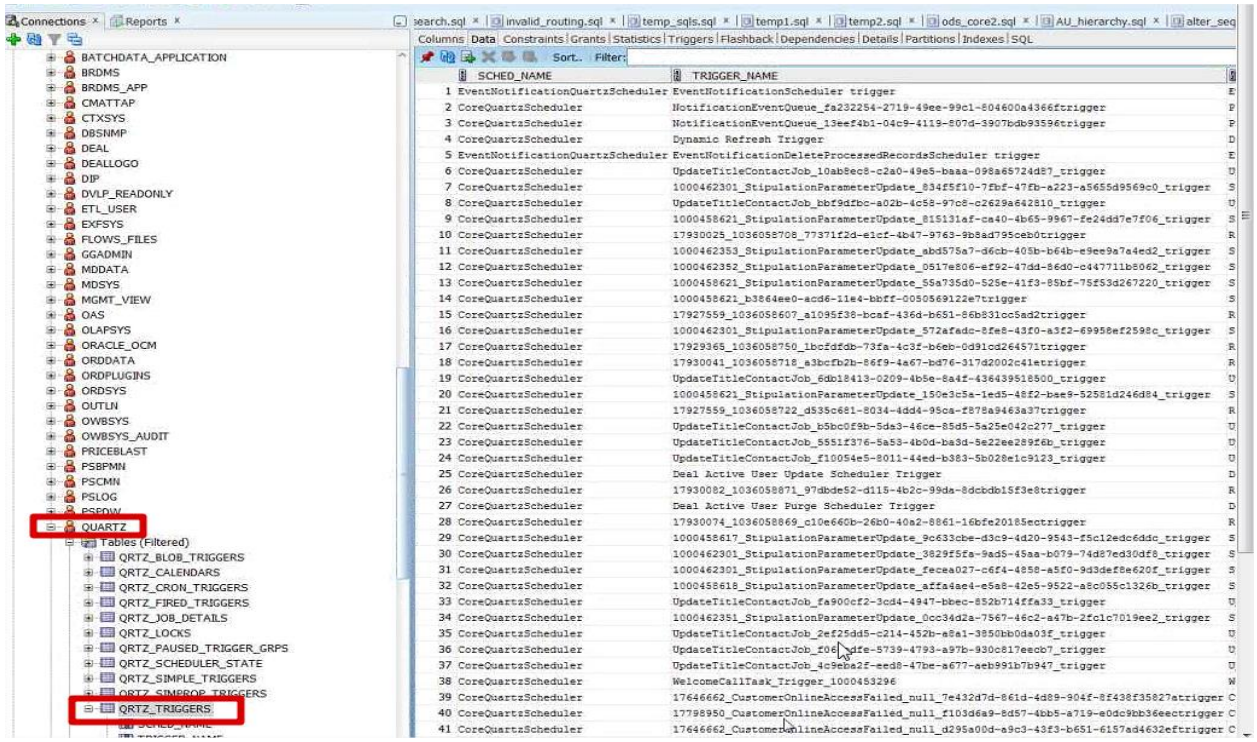
Quartz job picks up emails for every 15 mins and runs the email job.

<u>QUARTZ JOBS TABLES</u>

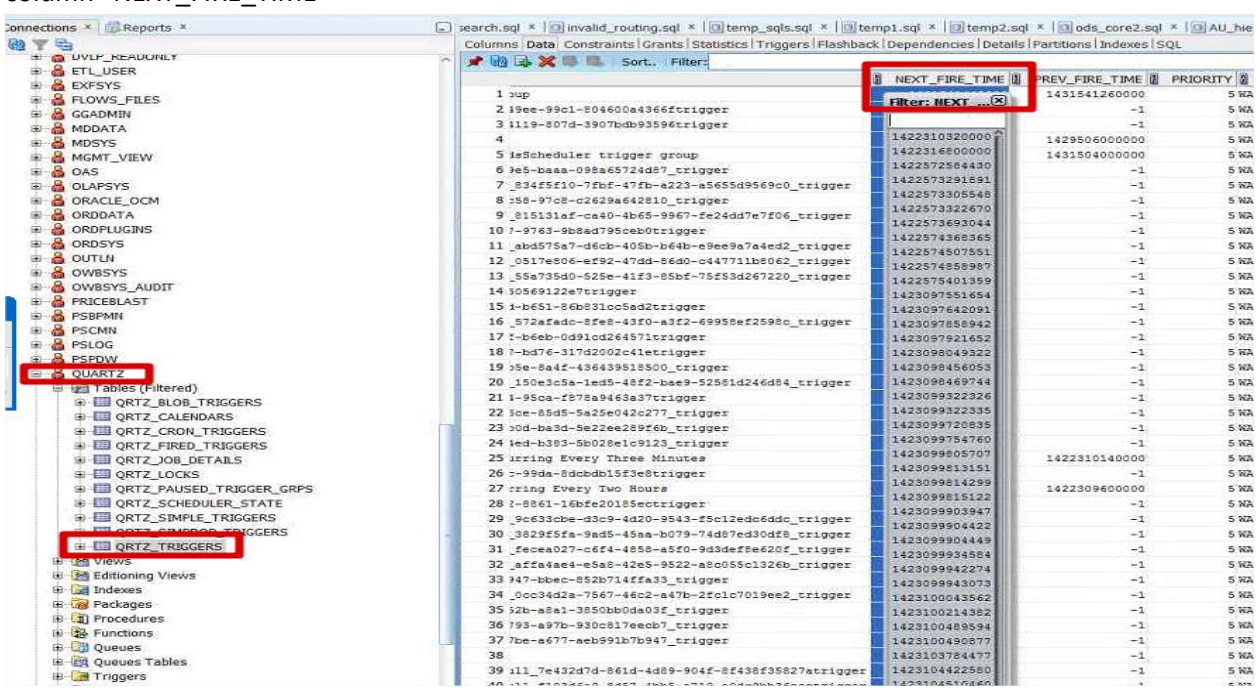
- We have separate schema in data base for managing the jobs, schedulers, triggers etc..
- The Schema name itself “QUARTZ”.



- QUARTZ_TRIGGERS is the key table and contains all the triggers, Scheduler Names, Job Name, Start time, End time etc.



- For each job, there should be each scheduler instance. Each job is supposed to run in each scheduler.
- In the cases where Server goes down or Job was down, Scheduler is smart enough to run the job again when it is available. Scheduler will restart all the jobs when the server came back up.
- If I want to re-run the job at specific time, I can go to QRTZ_TRIGGERS table and update the column “NEXT_FIRE_TIME”



Running Jobs locally:

- core-quartz.properties file is important. This file has the entire configuration to make the job to run locally. Since we don't have local database to manage/run the jobs. For local jobs, we configured to run in-memory. If we want to run local jobs, we should go for in memory. RAM job store uses in memory

Name	Date modified	Type	Size
concurrency.properties	5/12/2015 11:32 PM	PROPERTIES File	
core-quartz.properties	5/12/2015 11:32 PM	PROPERTIES File	
coretools.properties	5/12/2015 11:32 PM	PROPERTIES File	
DataChangeDisplay.properties	5/12/2015 11:32 PM	PROPERTIES File	
deal-quartz.properties	5/12/2015 11:32 PM	PROPERTIES File	
DeliveryQueries.properties	5/12/2015 11:32 PM	PROPERTIES File	

core-quartz.properties

```
10 org.quartz.scheduler.instanceName = CoreQuartzScheduler
11 org.quartz.scheduler.instanceId = AUTO
12 #org.quartz.scheduler.batchTriggerAcquisitionMaxCount = 40
13 #org.quartz.scheduler.batchTriggerAcquisitionFireAheadTimeWindow = 300000
14 #org.quartz.jobStore.acquireTriggersWithinLock = true
15 #===============================================================================
16 # Configure ThreadPool
17 #===============================================================================
18 org.quartz.threadPool.class = org.quartz.simpl.SimpleThreadPool
19 org.quartz.threadPool.threadCount = 10
20 org.quartz.threadPool.threadPriority = 5
21 #===============================================================================
22 # Configure JobStore
23 #===============================================================================
24 org.quartz.jobStore.misfireThreshold = 3540000
25 org.quartz.jobStore.class = org.quartz.simpl.RAMJobStore
26 #org.quartz.jobStore.driverDelegateClass = org.quartz.impl.jdbcjobstore.StdJDBCDelegate
```

- Setting needs to be done to run the job local: **CoreActivatorImpl-> localQuartzEnableStatus = "true"** to run the job local.

*CoreActiva... | NewQuartzSc... | QuartzSched... | AsyncProces... | QuartzDelega...

```
120     @Autowired
121     protected RegisterEventListnerService registerEventListnerService;
122
123
124     private static Logger LOG = Logger.getLogger(CoreActivatorImpl.class);
125
126
127     private long startTime = 0;
128
129     private String localquartzEnableStatus = "true";
130
```

New Jobs:

- How do I know what scheduler to use for my new job?
  We do that through configuration. **QuartzSchedulerNameJobGroupConfiguration.xml**. This configuration file has all the info about schedulers etc..

```
core-quartz.properties   QuartzSchedulerNameJobGroupConfiguration.xml
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
3  <properties>
4    <comment>To run quartz job using specific scheduler instance based on job group(key)</comment>
5
6    <!--  Entries for AuditQuartzScheduler Begin
7     -->
8     <entry key="AuditSchedulerGroup">AuditQuartzScheduler</entry>
9     <entry key="AuditRecycleSchedulerGroup">AuditQuartzScheduler</entry>
10    <entry key="AuditDeleteProcessedRecordsSchedulerGroup">AuditQuartzScheduler</entry>
11    <entry key="NEW_JOB_Process Audit Changes">AuditQuartzScheduler</entry>
12
13   <!--  Entries for AuditQuartzScheduler End
14    -->
15   <!--  Entries for EventNotificationQuartzScheduler Begin
16    -->
17    <entry key="EventNotificationSchedulerGroup">EventNotificationQuartzScheduler</entry>
18    <entry key="NEW_JOB_Process Event Notification">EventNotificationQuartzScheduler</entry>
19    <entry key="EventNotificationDeleteProcessedRecordsSchedulerGroup">EventNotificationQuartzScheduler</entry>
20   <!--  Entries for EventNotificationQuartzScheduler End
21    -->
22   <!--  Entries for DocGenQuartzScheduler Begin
23    -->
24    <entry key="DistributionStatusProcessing hourly Job">DocGenQuartzScheduler</entry>
25    <entry key="Document Generation Job Group">DocGenQuartzScheduler</entry>
26    <entry key="Initialization and Expere Request">DocGenQuartzScheduler</entry>
27    <entry key="RF Commit Request">DocGenQuartzScheduler</entry>
28    <entry key="Distribution Request">DocGenQuartzScheduler</entry>
29    <entry key="NAS Cleanup Request">DocGenQuartzScheduler</entry>
30    <entry key="ExternalOrderRequestCOMBODIST">DocGenQuartzScheduler</entry>
31    <entry key="SchedulePricingCall">DocGenQuartzScheduler</entry>
```

Logs:

- Quartz logs will be on quartz instances.DEV8- 803 is a QUARTZ_SERVER.
- All the BPM's runs on WAS servers.

Possible Issues:

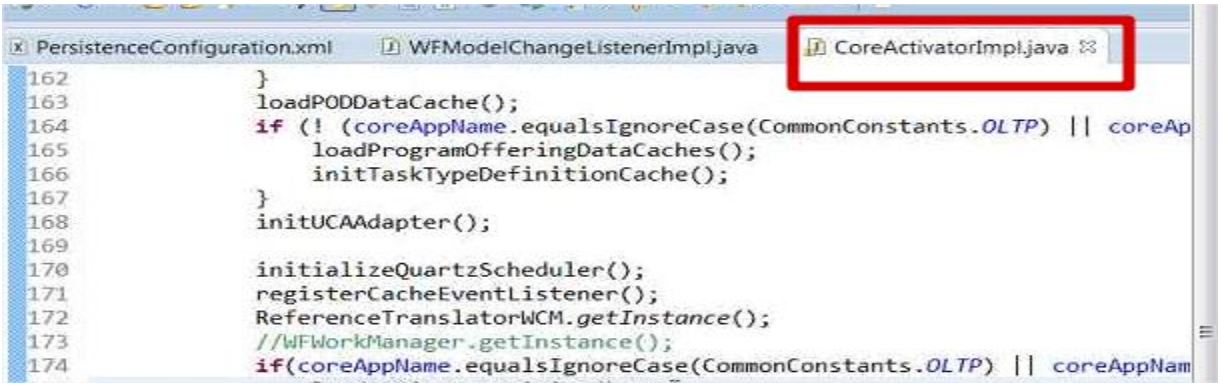
- Jobs configured to run at 2.00, ran at 8.00 AM.
- Most of the time, the above issue might be because of server was down.

CRON JOBS:

- CRON Jobs are for processing repeating tasks at regular intervals.
- These can be scheduled in DB with in QRTZ_CRON_TRIGGERS table.
- CRON jobs get started after server gets started.
- Between Simple vs CRON jobs- it's just a flag that makes the difference.
- The cron jobs are used when the jobs needs to be repeated. These jobs are not removed from DB as long as there is a future execution scheduled. We typically create cron jobs during server startup.

Important Points:

- During server start up itself, we start quartz scheduler.
- CoreActivatorImpl.java is the key program with respect to Quartz jobs.



- We are using plain Quartz for the jobs, but not the Spring-Quartz.
- For all Jboss schedulers, we initialize all the 7 properties files. Properties are nothing but, quartz properties for initialization start up.

- Below is the code where, we specify the properties for each high level Quartz job.

```
PersistenceConfiguration.xml   WFModelChangeListenerImpl.java   *CoreActivatorImpl.java   AsyncProcessSchedulerImpl.java
45    public static List<String> qrtzSchedulerList;
46    public static List<String> oltpQrtzSchedulerList;
47    public static List<String> auditQrtzSchedulerList;
48
49    static {
50        auditQrtzSchedulerList = new ArrayList<String>();
51        auditQrtzSchedulerList.add("audit-quartz.properties");
52
53        qrtzSchedulerList = new ArrayList<String>();
54        qrtzSchedulerList.add("core-quartz.properties");
55        qrtzSchedulerList.add("docgen-quartz.properties");
56        qrtzSchedulerList.add("stipulation-quartz.properties");
57        qrtzSchedulerList.add("eventNotification-quartz.properties");
58        qrtzSchedulerList.add("icmp-quartz.properties");
59        qrtzSchedulerList.add("eventCapture-quartz.properties");
60        oltpQrtzSchedulerList = new ArrayList<String>();
61        oltpQrtzSchedulerList.add("deal-quartz.properties");
62
63    }
64
```

- Below is the Initialization part of Quartz jobs:

```
Quick Access   Java EE   D
PersistenceConfiguration.xml   WFModelChangeListenerImpl.java   *CoreActivatorImpl.java   AsyncProcessSchedulerImpl.java
533            }
534        }
535
536    }
537
538    private void initializeQuartzScheduler() throws Exception {
539        PerfLogTimer performanceLogTimer = new PerfLogTimer();
540        try {
541            if (coreAppName.equalsIgnoreCase(CommonConstants.CORE_QUARTZ_APP) ||
542                (coreAppName.equalsIgnoreCase(CommonConstants.LOCAL) && localquartzEnableStatus != null && localquartzEnab
543                LOG.error("Info: Initializing QuartzScheduler...");
544
545                //create and start the schedulers
546                AsyncProcessSchedulerImpl.startAsyncProcessScheduler(true,AsyncProcessSchedulerImpl.qrtzSchedulerList);
547
548                if (!coreAppName.equalsIgnoreCase(CommonConstants.AUDIT)) {
549
550                    //NOTE: Old way of creating schedulers - Not Recommended.
551                    if (isProcessEnabled()) {
```

- We can stop specific job with single property change, with in the code.
- Quartz is available in all the nodes. So scheduling can be done on any nodes(JBoss).
- Quartz is initialized in all the servers(including OLTP) but only started on QUARTZ/TIER2 cluster.
- For some reason, OLTP is up but QUARTZ/TIER2 is down then the job is scheduled but executed only when Quartz is started.
- Same is true when QUARTZ/TIER2 is up but Quartz scheduler is stopped/paused.
- In a normal scenario for a simple job, The job is executed almost immediately and removed from DB once executed if the job is scheduled to run with current timestamp. The jobs scheduled for future execution are retained until the execution timestamp.
- When we display schedules in UI, we are trying to get node info from JVM. We go to JVM to get the status of Instance that running the job, not from DB. For this, we are using EJB's.
- We initially used EJB remote calls to get the status of node and node status. But recently they replaced EJB calls with HTTP requests

```
PersistenceConfiguration.xml   WFModelChangeListenerImpl.java   *CoreActivatorImpl.java   NewQuartzSchedulerBusinessServiceImpl.java
775             appliedUrl = quartzUrls;
776         }
777         String parameters = buildParmeterString(request);
778         for(String url : appliedUrl){
779             try {
780                 Request httpReq = HttpFactory.INSTANCE.createRequest();
781                 httpReq.setHttpMethod(HttpMethod.POST);
782
783                 String urlWithParams = url + "/servlet/quartzSchedulerServlet" + "?" + parameters;
784                 httpReq.setUrl(urlWithParams);
785                 httpReq.setRequestHeader("Content-type", "application/json");
786                 //String json = JSONUtil.convertToJSON(request);
787                 //HttpReq.setRequestBody(json); // server side does not support to read request content at this time
788                 Response response = httpSrv.execute(httpReq);
789                 if(response!=null && StringUtil.notEmpty(response.getResponseBody())){
790                 QuartzSchedulerResponse  result = JSONUtil.convertJSONToObject(response.getResponseBody(), QuartzSchedulerResponse.class);
791                 if(request.getUserName()!= null){
792                     result.setEnvironmentAdministrator(isEnvironmentAdmin(request.getUserName()));
793                 }
794                 result.setTargetUrl(url);
795                 retResponseList.add(result);
796                 }
797             } catch (Exception e) {
798                 LOG.error("Error during making request to ["+url+"]", e);
799             }
800         }
801         return retResponseList;
802     }
```

- **QuartzSchedulerBusinessServiceImpl.java** is also main class- **lookupSchedulerList** method is one of the key methods. If we look at **getAsyncProcessScheduler**()- instance name gets populated from this method.
- Get the URL list from Properties files and loop over the list of URL's. But the idea is to read from JVM.

```
WFModelChang...   *CoreActiva...   NewQuartzSc...   QuartzSched...   AsyncProces...   QuartzDelega...   AsyncProces...   AsyncProces...
828         if(request.getSchedulerStatus() != null){
829             params.append("&schedulerStatus=" + request.getSchedulerStatus());
830         }
831         if(request.getSchedulerInstanceId() != null){
832             params.append("&schedulerInstanceId=" + request.getSchedulerInstanceId());
833         }
834         if(request.getGroupStatus() != null){
835             params.append("&groupStatus=" + request.getGroupStatus());
836         }
837         if(request.getListOfGrps() != null){
838             params.append("&listOfGrps=" + listToString(request.getListOfGrps()));
839         }
840         return params.toString().substring(1).replace(" ", "%20");
841     }
842     private  HttpService getHttpService() {
843         if (httpService == null) {
844             HttpServiceUtil httpUtil = new HttpServiceUtil();
845
846             HttpServiceConfig httpServiceConfig = HttpFactory.INSTANCE.createHttpService();
847             httpServiceConfig.setConnectionTimeout(httpUtil.getTimeout());
848             httpServiceConfig.setMaxTotalConnections(httpUtil.getMaxTotalConnections());
849             httpServiceConfig.setDefaultMaxConnectionsPerHost(httpUtil.getMaxConnectionsPerHost());
850             httpServiceConfig.setStaleCheckingEnabled(httpUtil.isConnectionStaleCheckingEnabled());
851
852             HttpService service = new HttpServiceImpl();
853             service.setServiceId("QuartzMgtService");
854             httpService = service;
855             httpService.setHttpServiceConfig(httpServiceConfig);
856         }
```

```
PersistenceConfiguration.xml   WFModelChangeListenerImpl.java   *CoreActivatorImpl.java   NewQuartzSchedulerBusinessServiceImpl.java
760             }
761         }
762         return schedulerTypeProperty;
763     }
764
765     @Override
766     public List<QuartzSchedulerResponse> processRequest(QuartzSchedulerRequest request) {
767         List<QuartzSchedulerResponse> retResponseList = new ArrayList<QuartzSchedulerResponse>();
768         getQuartzUrls();
769         HttpService httpSrv = getHttpService();
770         List<String> appliedUrl = null;
771         if(request.getTargetUrl() != null) {
772             appliedUrl = new ArrayList<String>();
773             appliedUrl.add(request.getTargetUrl());
774         } else {
775             appliedUrl = quartzUrls;
776         }
777         String parameters = buildParmeterString(request);
778         for(String url : appliedUrl){
779             try {
780                 Request httpReq = HttpFactory.INSTANCE.createRequest();
781                 httpReq.setHttpMethod(HttpMethod.POST);
782
783                 String urlWithParams = url + "/servlet/quartzSchedulerServlet" + "?" + parameters;
784                 httpReq.setUrl(urlWithParams);
785                 httpReq.setRequestHeader("Content-type", "application/json");
786
```

- All the jobs we start during server start up is **CRON** jobs.
- Only emailing is STATEFUL job. All the other jobs are stateless.

Key Service:

```
WFModelChang...   *CoreActiva...   NewQuartzSc...   QuartzSched...   AsyncProces...   QuartzDelega...   AsyncProces...   AsyncProces...
1  package com.wellsfargo.core.sharedlib.delegator.appadmin;
2
3
4  public interface AsyncProcessSchedulerLocal {
5
6      public String getQuartzUrls();
7
8      public  String getSchedulerList(String schedulerType) ;
9
10     public  void changeSchedulerStatus(String jsonString);
11
12     public  void pauseAllSchedulers(String jsonString);
13
14     public  void interruptSchedulerJob(String jsonString);
15
16
17     //Quartz Scheduler production support tool
18     public String loadSchedulerGroups(String jsonString);
19     public String loadJobsInAGroups(String jsonString);
20     public String loadGroupsInAScheuler(String jsonString);
21     public void pauseListOfSeletedGrps(String jsonString);
22     public void resumeListOfSeletedGrps(String jsonString);
23     public void pauseListOfSeletedJobs(String jsonString);
24     public void resumeListOfSeletedJobs(String jsonString);
25     public void unScheduleListOfSeletedJobs(String jsonString);
26     public void reScheduleListOfSeletedJobs(String jsonString);
27     public String processRequest(String jsonString);
28 }
```

Creating Job Params:

```
*CoreActiva...  NewQuartzSc...  QuartzSched...  AsyncProces...  QuartzDelega...  AsyncProces...  AsyncProces...  RefreshDynam...
29          AsyncProcessConstants.REFRESH_DYNAMIC_GROUP_JOB_NAME, AsyncProcessConstants.REFRESH_DYNAMIC_GROUP_NAME, null);
30      AsyncProcessRequest aProcReq = new AsyncProcessRequest();
31      aSender.sendMessage(aProcReq, createJobParams());
32  } catch (Throwable t) {
33      logger.error("RefreshDynamicGroupsSchedular ::: ERROR:::" + t.getMessage());
34  }
35  }
36
37  private static AsyncProcessSchedulerJobParams createJobParams() {
38  ResourceBundle props = ResourceBundle.getBundle("core-quartz");
39  String daily705amCronExpr = null;
40  if (props != null) {
41      daily705amCronExpr = props.getString("cron.expression.scheduler.daily.705am");
42  }
43  AsyncProcessSchedulerJobParams jobParams = new AsyncProcessSchedulerJobParams();
44  jobParams.setJobName(AsyncProcessConstants.REFRESH_DYNAMIC_GROUP_JOB_NAME);
45  jobParams.setGroupName(AsyncProcessConstants.REFRESH_DYNAMIC_GROUP_NAME);
46  jobParams.setCronExpr(daily705amCronExpr);
47  jobParams.setTriggerName(AsyncProcessConstants.REFRESH_DYNAMIC_GROUP_TRIGGER_NAME);
48  jobParams.setTriggerDesc(AsyncProcessConstants.REFRESH_DYNAMIC_GROUP_NAME);
49  jobParams.setSimpleTrigger(false);
50  jobParams.setCalendarName(AsyncProcessConstants.CAL_NAME);
51  jobParams.setHandlerName("com.wellsfargo.core.business.service.dp.task.RefreshDynamicGroupsHandlerImpl");
52  jobParams.setClassName(AsyncProcessStatelessJobToRun.class);
53
54  return jobParams;
55  }
56
```

Sample Quartz call from Listener:

```
AsyncProces...  *RefreshDyna...  AsyncProces...  AsyncProces...  Distributio...  QuartzSched...  WelcomeCall...  MilestoneLi...
225                 if (MilestoneStatusEnum.POST_FUNDS_DISBURSED_MODIFICATION_COMPLETE.getCode().equals(milestone.getStatus(
226                     uwDecisionSummaryBusinessService.generateUDSfromMilestone(deal.getEnterpriseDealNumber(), "UW Decisic
227                 }
228
229                 String fulfillmentLob=lineOfBusinessService.getFulfillmentLobByDeal(deal.getId());
230                 lob = fulfillmentLob != null ? fulfillmentLob : lineOfBusinessService.getSalesLobByDeal(deal.getId());
231                 //logging for testing
232                 if (logger.isDebugEnabled()) {
233                     logger.debug("Retrieved lob for dealId "+deal.getId()+"-"+lob);
234
235                  }
236                 if (MilestoneStatusEnum.SUBMITTED.getCode().equals(milestone.getStatus().getCode())) {  omDeploymentProp
237
241                     if (logger.isDebugEnabled()) {
242                         logger.debug("Retrieved scheduleTime for dealId"+deal.getId()+" lob RTL"+":"+welcomeCallSche
243
244                     }
245
246                     WelcomeCallPriorityEscalteScheduler.triggerQuartzJob(welcomeCallScheduledTime,deal.getId());
247                 }
248
```

LISTENERS:

- Whenever any events occurred on specific columns on a specific table, the listeners gets fired and will process the logic.
- All the configuration for listeners is under : **PersistenceConfiguration.xml**
- Listeners with Priority = 0 -> highest priority. Listeners with 0 will be executed first.
-
- Listeners with Priority = 100 -> lowest priority

```
serviceInterface="com.wellsfargo.core.extended.entity.listener.audit.WFModelChangeListener" priority="100"
```

```
serviceInterface="com.wellsfargo.core.extended.entity.listener.stipulation.StipulationDataChangeListener" priority="95"
```

Key Listener: WFModelChangeListener

```
serviceInterface="com.wellsfargo.core.extended.entity.listener.audit.WFModelChangeListener"
```