System Administration - Background Job Processing Online Training

Abstract:

This training includes details of two job application - Executor and Scheduler and will show how job processing works, and what kind of business challenges you may need to address using this functionality.

Detailed objectives. After the training you will know:

- What are differences between Job Executor and Job Scheduler
- How they work and how to use all their features

Target audience:

- People who will need to design, implement, and/or maintain jobs in DELMIA Apriso solutions

Requirements:

- General understanding of Job Executing, DELMIA Apriso Portal Usage, and Windows Administration.
- Recommended (but not required) trainings:
  - ADM001 - System Administration Installation
  - ADM002 - System Administration Upgrade
  - ADM003 - System Administration Configuration

Role and level:

- DELMIA Apriso Administrators

Duration: 60 min

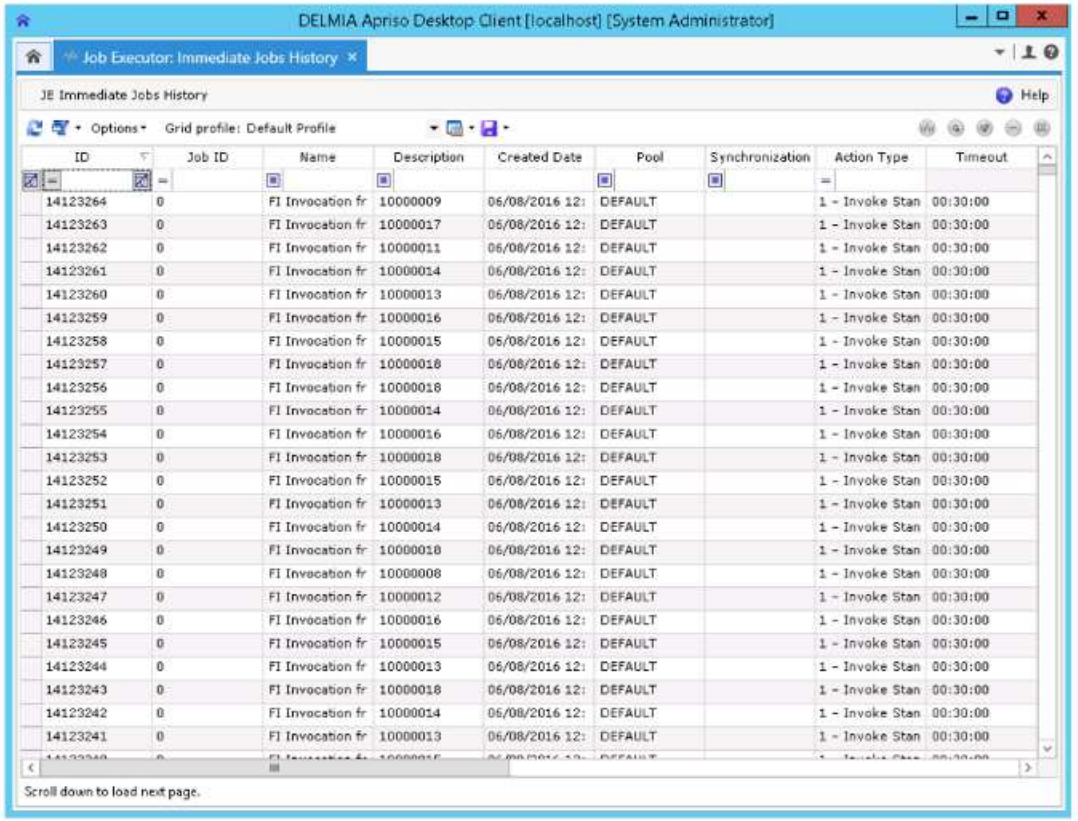
Job Executor (JE)

Job Executor (JE) is the primary DELMIA Apriso component for processing system background tasks. It is designed as a high-performance execution engine for executing queue-like (FIFO - First In, First Out) background tasks. Job Executor is complemented by a second service - Job Scheduler, which is optimized for processing recurring, scheduled jobs.

Job Executor consists of two parts: a background service that actually executes jobs and UI monitoring screens that allow for the monitoring, configuration and troubleshooting of Job Executor.

In JE includes some features that are not supported by JS, mainly:

- Processing in pools
- Prioritization of jobs
- Configurable execution attempts, their number, and sleep time
- Ability to run multiple instances of the service for maximum performance

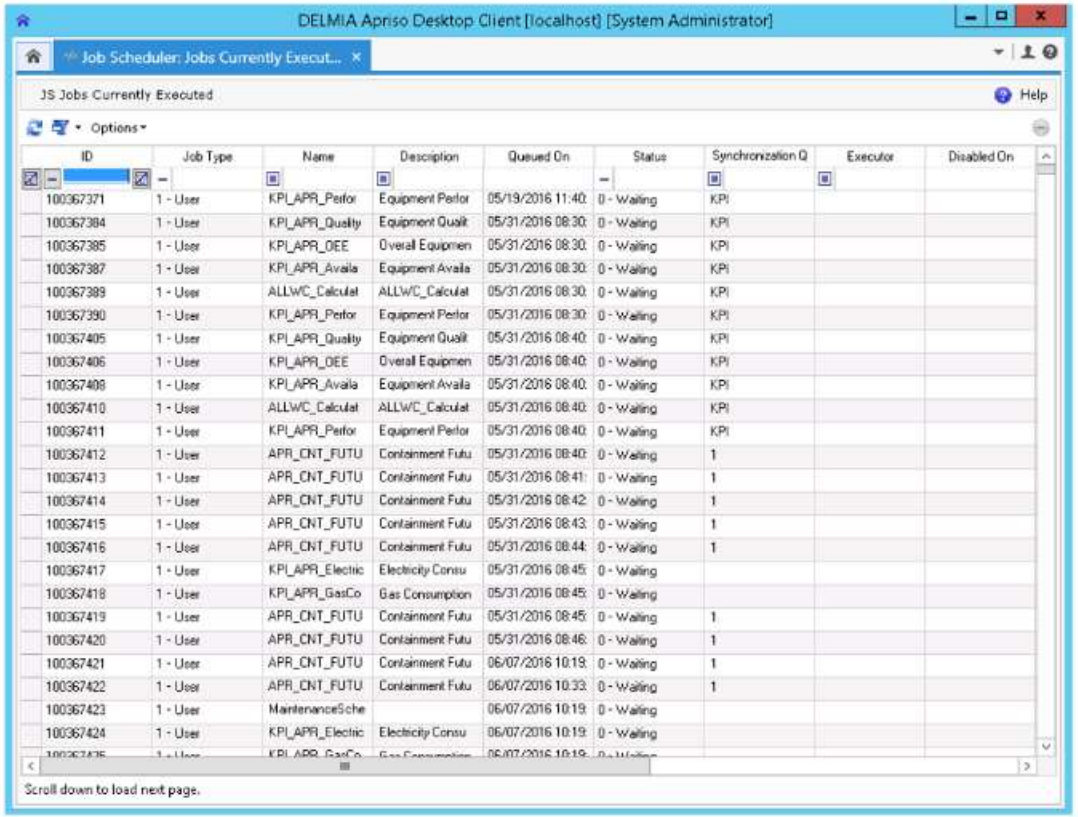


Job Scheduler (JS)

Job Scheduler (JS) is the secondary DELMIA Apriso component for processing system background tasks. It is designed as a flexible engine for processing complex recurring, scheduled jobs.

Job Scheduler consists of two parts: a background service that actually executes jobs and UI monitoring screens that allow for the monitoring, configuration and troubleshooting of Job Scheduler

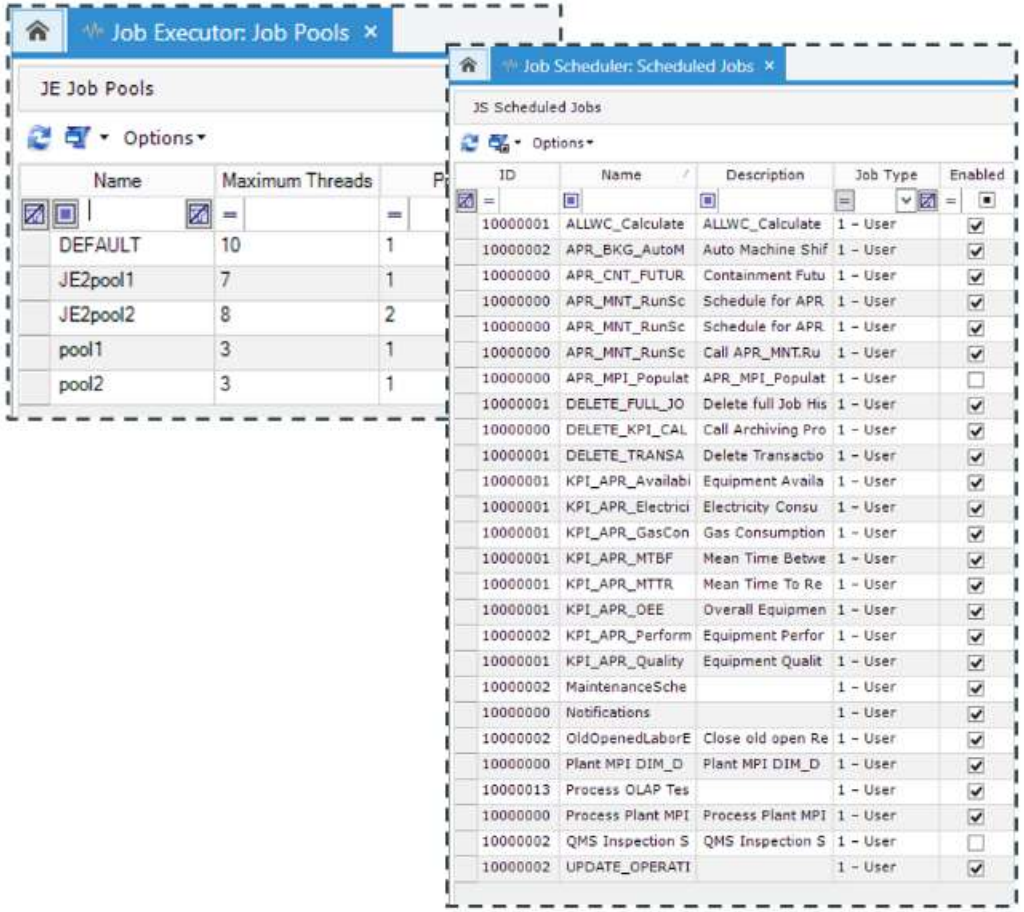
With JS, multiple actions per jobs can be processed, as well as multiple schedules per job. Also, there is an alarm service - JS will send alerts in the event of failed execution.



JE and JS Common Features

JE and JS share some common features:

- Monitoring and troubleshooting user interface
- Synchronization queues
- Monitoring screens
- Same model for action type and parameters configuration
- Automatic retry for jobs failed due to deadlock or concurrency error



Job Executor vs Job Scheduler

JE and JS also differ in many aspects:

Job Executor (JE)	Job Scheduler (JS)
Jobs for immediate execution.	Jobs scheduled for the future or having recurring schedule.
Large number of single, immediate queue-like (FIFO) jobs.	Small number of recurring jobs scheduled for multiple executions over a period of time (recurring).
Jobs with only one action, only a schedule to be run immediately.	Complex jobs (multiple actions, multiple schedules).
Jobs created by other DELMIA Apriso modules (Business Integrator, Printing, FI).	Jobs usually created manually by Administrators.
Reprocessing performed by creating new jobs (duplicating existing ones).	Reprocessing performed by creating new schedules for the same job.

Chapter 2: JE Processing Model

In this chapter there will be more details about Job Executor as the the primary DELMIA Apriso component for processing system background tasks.

Here are the chapters to be covered:

1. *Overview of Job Processing*
2. **JE Processing Model**
3. JS Processing Model

JE Processing Model, part 1

Job Executor is a performance-optimized background processing engine. It processes jobs in a queue-like fashion while respecting job pool priorities and thread limitations.

The following section describes details about the Job Executor processing model.

- JE periodically scans the JOB_QUEUE table and loads new jobs into memory
- Jobs are loaded in batches
- Both, the batch size and load frequency are configurable options of JE
- When new jobs are loaded from the database, they are grouped by a job pool
- JE keeps a list of pools with assigned jobs and scans the pools in the order of pool priority (there are no priorities on the job level) to find the first job to be executed

Database: Apriso
Table: JOB_QUEUE

JE Processing Model, part 2

Job Executor is a performance-optimized background processing engine. It processes jobs in a queue-like fashion while respecting job pool priorities and thread limitations.

The following section describes details about the Job Executor processing model.

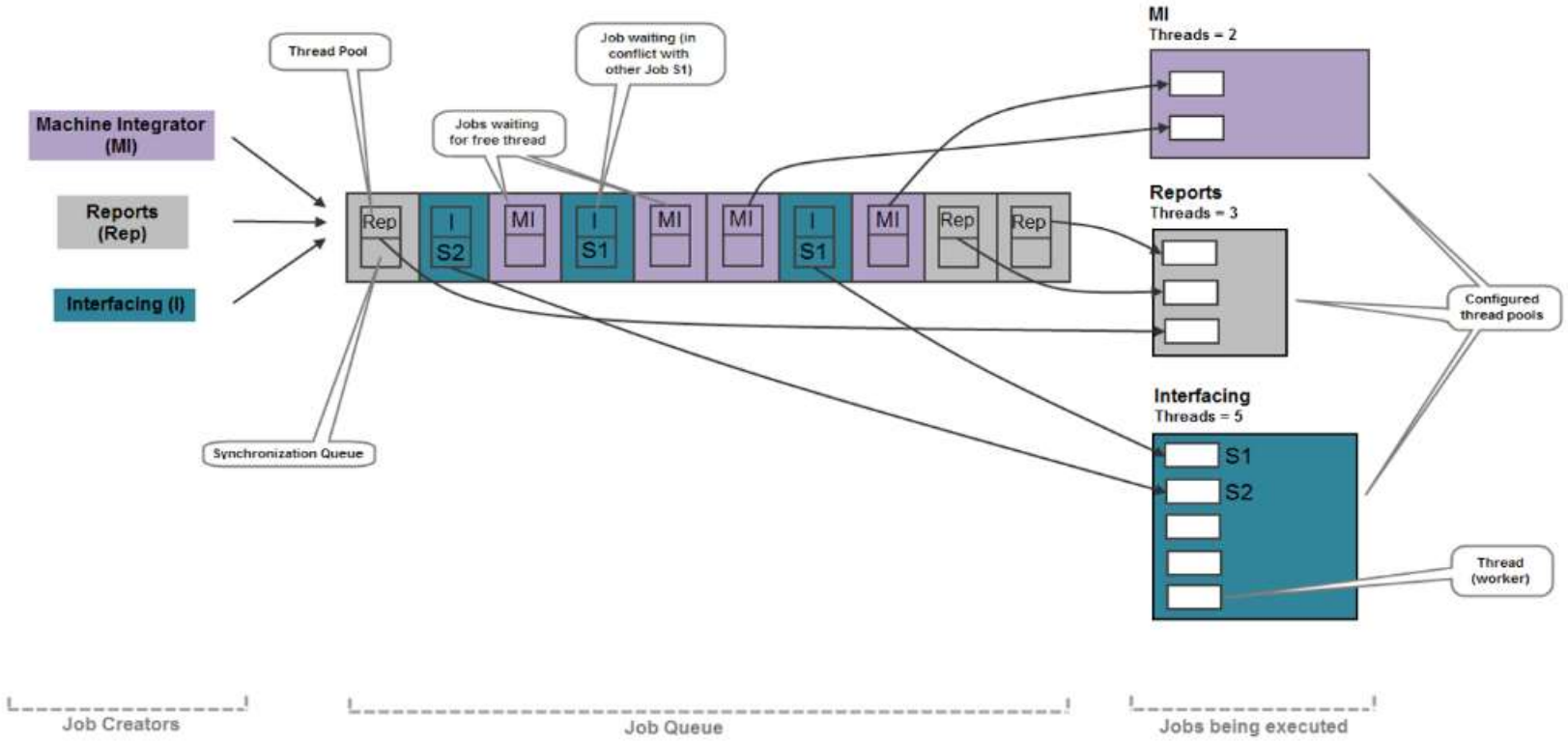
Database: Apriso
Table: JOB_QUEUE

- When such a job is located, JE checks the number of threads used by other jobs being executed from the same pool, and if the number is not bigger than the maximum number of threads for this pool, it starts the execution of the job
- If the maximum number of threads is exceeded, JE continues scanning for new jobs belonging to lower priority pools

JE Processing Model - Sequence

In cases where jobs must be executed in a particular order, Synchronization Queues should be used.

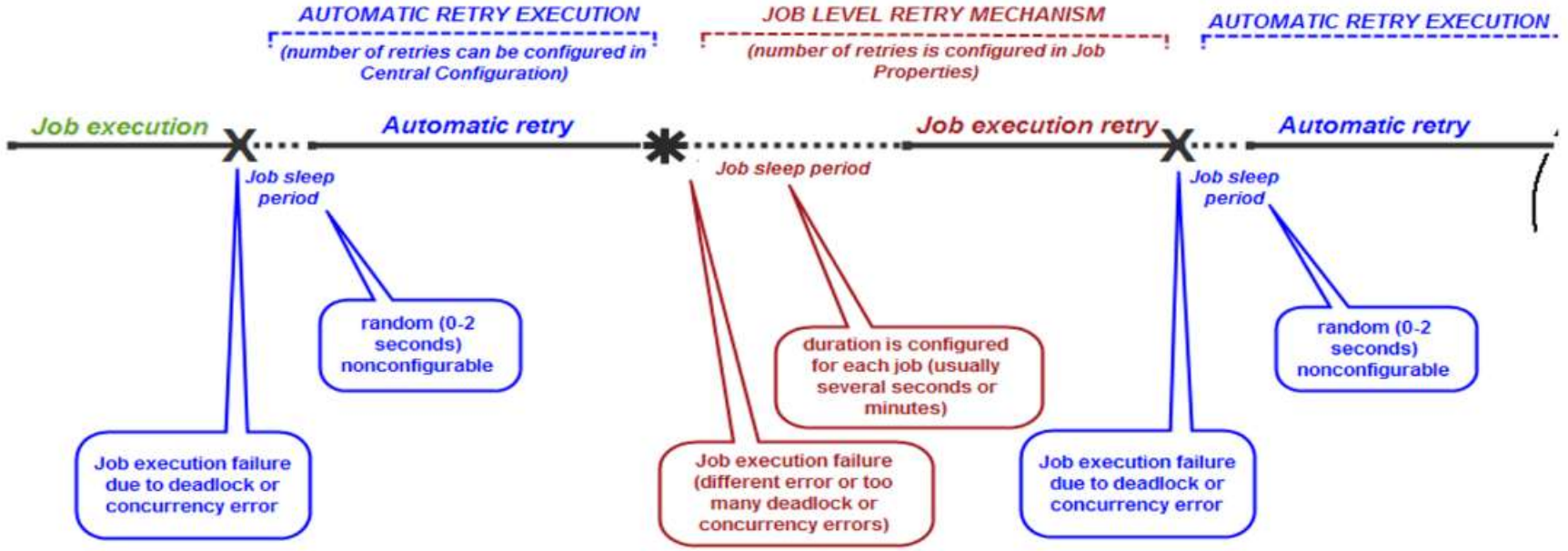
Each job can have a Synchronization Queue assigned, which is an identifier (string) that groups all jobs from the same pool that should be executed exactly in the same order as they are created.



JE Processing Model - Double Retry Mechanism

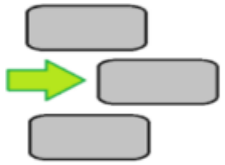
JE also contains a double retry mechanism:

- Automatic retry - in case of deadlock or concurrency error that causes Job failure (it can be configured by JE)
- Job level retry mechanism - each job can be configured to be reprocessed in case of any error

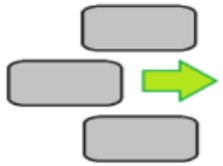


JE Processing Model - Finishing

- After a job execution is completed, the related row is deleted from the JOB_QUEUE table
- Then row is copied and inserted into the JOB_QUEUE_HISTORY table - it keeps a history of executions for all jobs together with the original job definition
- The JOB_QUEUE_HISTORY table should be periodically archived to remove unnecessary old data and avoid database overgrowth



Job Queue



Job Queue History

Job Queue History Archiving

JE Self-Monitoring

Memory management - this self-check is performed every 5 minutes and is controlled by

- Private Bytes performance counter, where the following memory thresholds apply:
  - For 32-bit environments - 1,5 GB
  - For 64-bit environments - 3 GB
    I. if the values above are exceeded, service restart is performed
- Database access control - if no database access is detected, the service remains in starting status until the connection is restored; after establishing the connection, the service proceeds to started status
- Job timeout check - the check is performed every 5 minutes; the service is restarted when the following condition is met:
  - Now - Job Timeout > Last Job Timeout + 20%

JE Recycling:

- In the case of database access problems, the service is configured and maintained by infrastructure/operating system and not by Job Executor
- In the case of failure:
  - Non-clustered environment - the behavior is controlled by Windows service properties
  - Clustered environment - several service restarts are performed (depending on cluster configuration); if the problem persists after the specified number of retries, it is switched to another node

JE Central Configuration

For proper configuration, it is mandatory to provide the proper data in the Central Configuration file with a path to system services:

```
<add key="SchedulingServices" value="REMOTING:net.tcp://${AppAddress}:32607/JobSchedulerService" />
<add key="JobExecutorServices" value="REMOTING:net.tcp://${AppAddress}:32612/jobexecutorservice" />
<add key="GlobalProcessManagerServices" value="REMOTING:net.tcp://${AppAddress}:32709/gpmservice" />
```

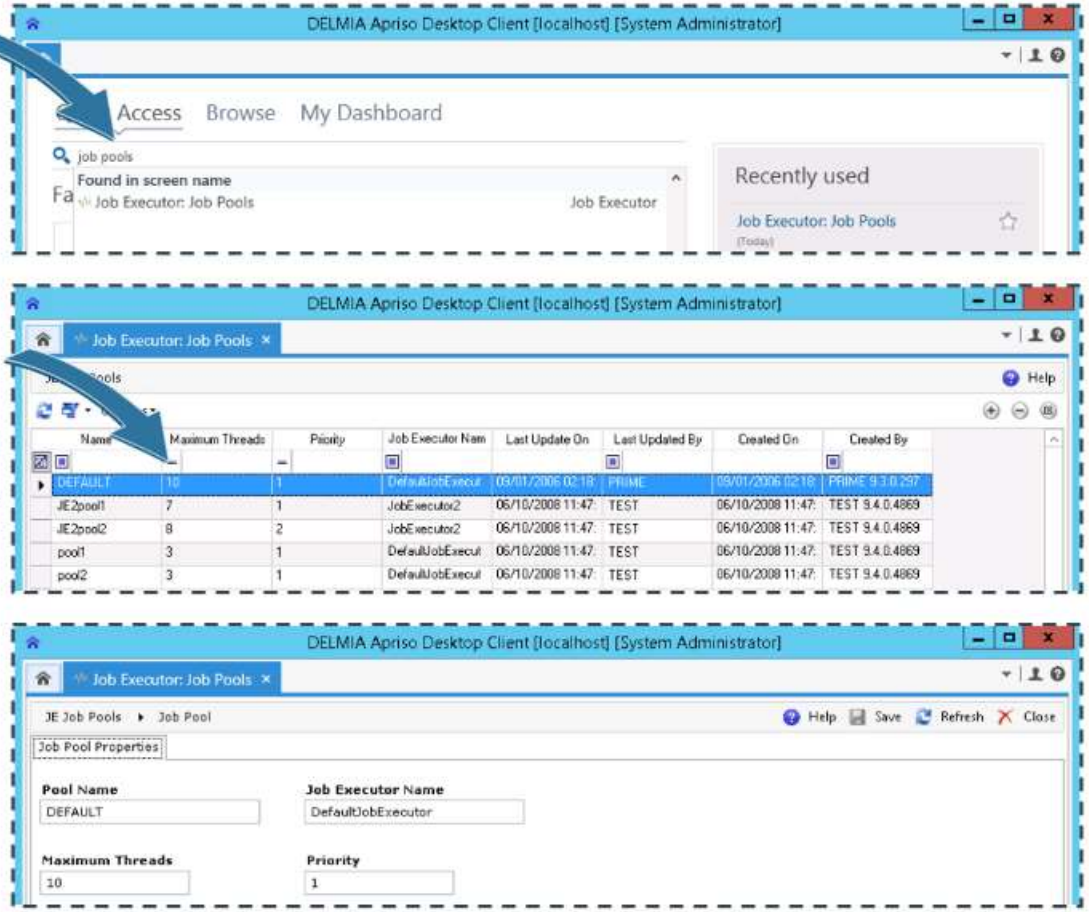
```
<FlexNet.JobExecutor>
  <add key="MaxExecutionThreads" value="10" />
  <add key="JobLoadRate" value="1000" />
  <add key="JobExecutorName" value="DefaultJobExecutor" />
  <add key="JobLoadBatchSize" value="50" />
  <add key="NumberOfAutomaticProcessingRetries" value="1" />
</FlexNet.JobExecutor>
```

Job Pool Configuration Screen

Every job processed by JE must be assigned to one of the configured job pools (job pools allow prioritization of job execution). Each pool contains the following configurable parameters:

- **Pool name** - User friendly name of the pool. (This name is used later to select from the list of available pools.)
- **JE Name** - Name of the JE instance that processes the job assigned to this pool. (Each JE instance takes its name from its configuration file during start-up.)
- **Maximum threads** - Number of maximum execution threads used to process jobs assigned to this pool. (This parameter allows limiting the processing power for the pool to allow other pools to be processed at the same time.)
- **Priority** - Priority of the pool. JE always tries to process jobs assigned to higher priority pools first, respecting the maximum number of threads utilized for the pool

All parameters can be configured using the Job Pools configuration screen.



Configuration of Job Pools

Configuration of job pools has an impact on the way jobs are processed and may influence the performance and responsiveness of JE

- JE can work with a default configuration that contains only one job pool (all jobs are processed in the same way, have equal priorities, and are executed as they come)
- It is recommended to define separate job pools for jobs requiring different priority of processing. By limiting the number of available threads it is possible to eliminate the potential issue of “starving” lower priority pools

Job Executor: Job Pools

JE Job Pools

Options

Name	Maximum Threads	Priority	Job Executor Nam	Last Update On	Last Updated By	Created On	Created By
DEFAULT	10	1	DefaultJobExecut	09/01/2006 05:18:	PRIME	09/01/2006 05:18:	PRIME 9.3.0.297
JE2pool1	7	1	JobExecutor2	06/10/2008 02:47:	TEST	06/10/2008 02:47:	TEST 9.4.0.4869
JE2pool2	8	2	JobExecutor2	06/10/2008 02:47:	TEST	06/10/2008 02:47:	TEST 9.4.0.4869
pool1	3	1	DefaultJobExecut	06/10/2008 02:47:	TEST	06/10/2008 02:47:	TEST 9.4.0.4869
pool2	3	1	DefaultJobExecut	06/10/2008 02:47:	TEST	06/10/2008 02:47:	TEST 9.4.0.4869

Sample Configuration of Job Pools - Correct

The configuration can be done in few way, but the mportant paramenter to consider is MaximumExecutionThreads.

Let's assume, that MaximumExecutionThreads is set as 10. Now, we can divide the slots in different ways:

Example 1: In this configuration "Printing" and "Default" are executed with the same priority. "Printing" is limited to a maximum of 3 concurrent jobs and "Default" is limited to 5. As a result, all pools can be processed at the same time. The "Interfacing" pool can take all threads only if there are no higher priority jobs. This is one of the correct ways.

Pool name	Priority	Threads
Printing	1	3
Default	1	5
Interfacing	3	10

Example 2: In this configuration, the highest priority belongs to "Machine Processing" jobs. They are executed first using a maximum of 2 threads. Remaining available threads (8) are split between "Print" and "Default" and "Interfacing" jobs. This means that all job pools can be executed at the same time, if required. The "Interfacing" pool can take all threads only if there are no higher priority jobs. This also allocates the slots correcntly.

Pool name	Priority	Threads
Machine Processing	1	2
Printing	2	3
Default	3	3
Interfacing	4	10

Sample Configuration of Job Pools - Incorrect

Example 3: The below table shows that all possible threads can be taken by the “Printing” pool. This may result in the “Default” and “Interfacing” pools being “starved” and not processed until all “Printing” jobs are completed. This is not recommended, as it doesn’t use the slots efficiently.

Pool name	Priority	Threads
Printing	1	10
Default	2	5
Interfacing	3	10

Example 4: In the table below, all possible (10) threads are not distributed. This means that each job pool is processed in the one-job-at-the-time mode. Increasing the number of threads could improve performance. Again, the available slots are waisted in this example.

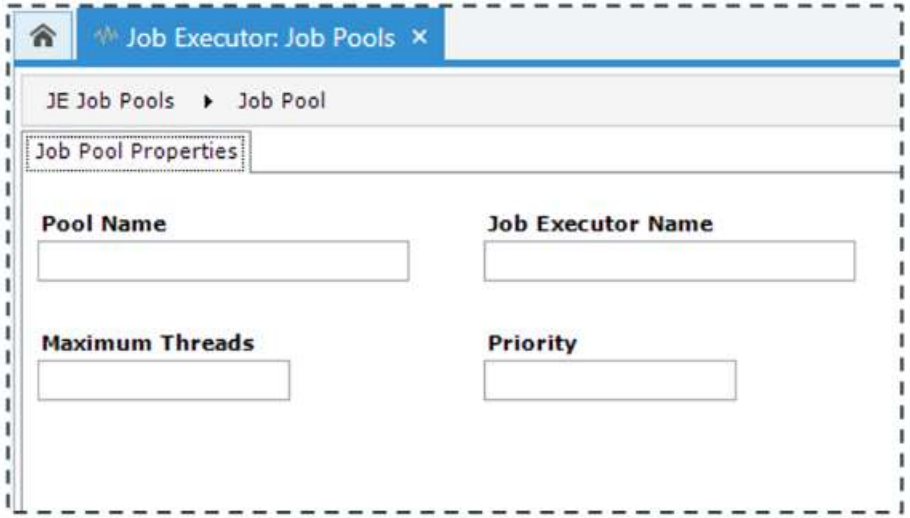
Pool name	Priority	Threads
Printing	1	1
MI Events	2	1
Default	3	1
Interfacing	4	1

The configuration of job pools requires knowledge about the type of processing expected in the given deployment of DELMIA Apriso. It may also require some experimenting and performance tuning to obtain the best results.

Job Pools - Recommendations

There are some recomendations for Job Pools. Proper configuration of job pools ensures smooth execution of jobs

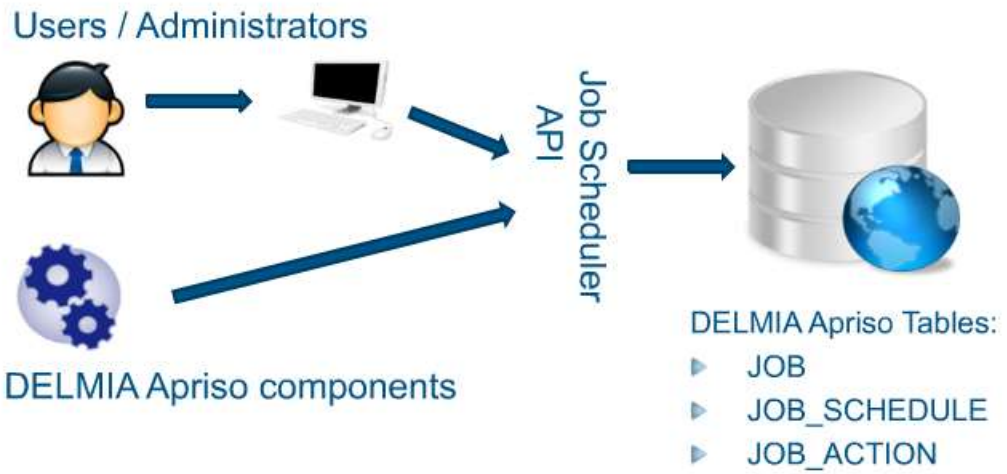
- Create several job pools based on different job types
- Assign priorities based on requirements
- Do not assign all threads to a single pool
- Consider the threads as workers:
  - Assign maximum number of workers to a given pool
  - Workers are shared by the pools



JS Processing Model

Job Scheduler is designed for processing jobs that should be executed at a specific point in time. Each job can be scheduled for a particular moment in time or can have a recurring schedule. It is best used for recurring administration-like tasks.

- JS periodically scans the JOB table and loads new jobs into memory
- Load frequency is a configurable option of JS
- JS can also be notified by a calling component that a new job is created so that JS can load the job without waiting for a full period
- New jobs are loaded into memory, then JS calculates the nearest execution time for each job based on job schedules
- When execution time for the job comes, JS publishes the job for execution by inserting it into JOB_EXECUTION_STATUS table
- The second subsystem of JS loads the action from JOB_EXECUTION_STATUS table and starts the actual execution of the actions using all available threads



JS Processing Model - Retry

- JS also supports automatic retry of an action execution when it fails due to database deadlock or concurrency error
- Each job is executed as a single transaction. If the job execution fails, the whole transaction is rolled back
- After the job has completed, JS stores history of the execution in tables JOB_HISTORY and JOB_ACTION_HISTORY. Each execution of the job adds new rows into these tables
- All history rows are related to the single job stored in the JOB table

JS Central Configuration

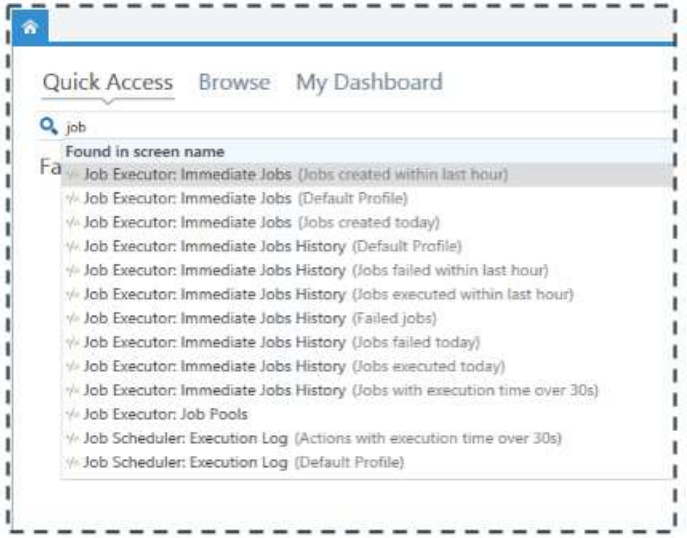
For proper configuration, it is mandatory to provide the proper data in the Central Configuration file with a path to system services:

```
<add key="SequenceProviderServices" value="REMOTING:net.tcp://${AppAddress}:32601/frameworkservice" />
<add key="SchedulingServices" value="REMOTING:net.tcp://${AppAddress}:32607/JobSchedulerService" />
```

```
<FlexNet.Scheduling>
  <add key="ConfigurationLoaderSleepPeriod" value="300" />
  <add key="NumberOfExecutorThreads" value="5" />
  <add key="MinimalNotificationPeriod" value="10" />
  <add key="NumberOfAutomaticProcessingRetries" value="1" />
  <add key="EnableMaintenanceOldSchedule" value="false" />
</FlexNet.Scheduling>
```

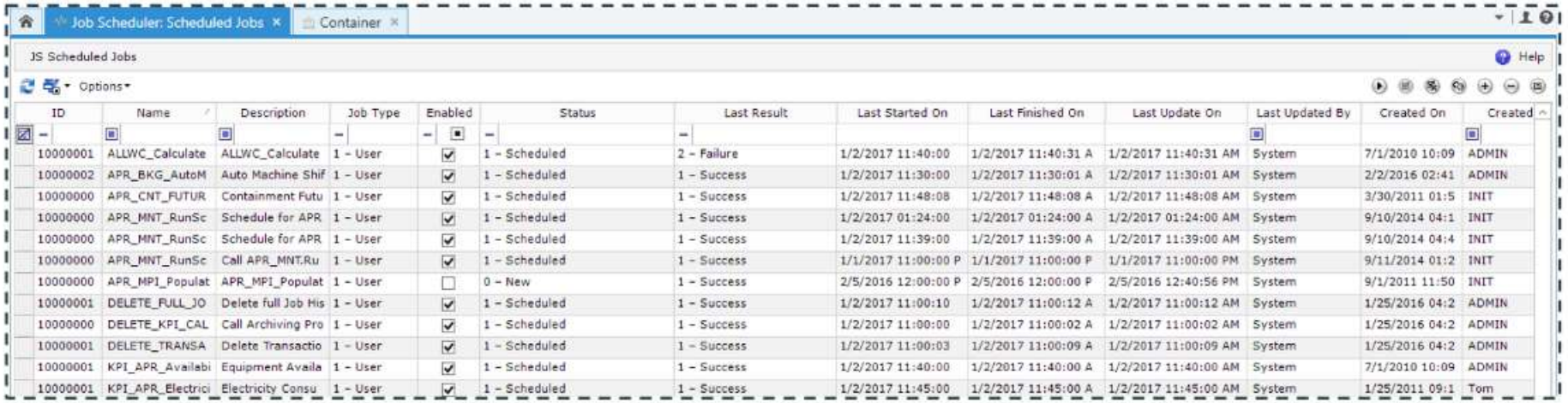
Job Executor and Scheduler User Interface

- Scheduled Jobs
- Jobs Currently Executed
- Synchronization Queues Status
- XML Messages
- Execution Log
- Job Pools
- Immediate Jobs
- Immediate Jobs History



Scheduled Jobs

- Used by JS
- Allows creation of user defined jobs:
- Defines job schedules
- Defines job actions
- Displays all user defined jobs
- Enables and disables jobs
- Executes jobs manually:
- Does not need a schedule to do this
- Does not display jobs related to XML Messages originating from BI

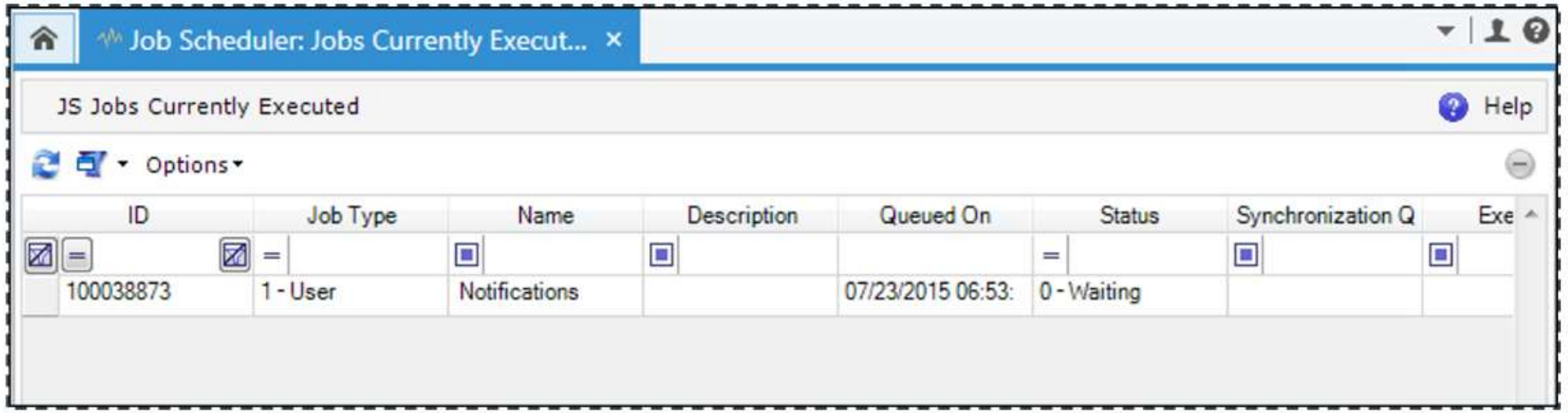


ID	Name	Description	Job Type	Enabled	Status	Last Result	Last Started On	Last Finished On	Last Update On	Last Updated By	Created On	Created
10000001	ALLWC_Calculate	ALLWC_Calculate	1 - User	☑	1 - Scheduled	2 - Failure	1/2/2017 11:40:00	1/2/2017 11:40:31 A	1/2/2017 11:40:31 AM	System	7/1/2010 10:09	ADMIN
10000002	APR_BKG_AutoM	Auto Machine Shif	1 - User	☑	1 - Scheduled	1 - Success	1/2/2017 11:30:00	1/2/2017 11:30:01 A	1/2/2017 11:30:01 AM	System	2/2/2016 02:41	ADMIN
10000000	APR_CNT_FUTUR	Containment Futu	1 - User	☑	1 - Scheduled	1 - Success	1/2/2017 11:48:08	1/2/2017 11:48:08 A	1/2/2017 11:48:08 AM	System	3/30/2011 01:5	INIT
10000000	APR_MNT_RunSc	Schedule for APR	1 - User	☑	1 - Scheduled	1 - Success	1/2/2017 01:24:00	1/2/2017 01:24:00 A	1/2/2017 01:24:00 AM	System	9/10/2014 04:1	INIT
10000000	APR_MNT_RunSc	Schedule for APR	1 - User	☑	1 - Scheduled	1 - Success	1/2/2017 11:39:00	1/2/2017 11:39:00 A	1/2/2017 11:39:00 AM	System	9/10/2014 04:4	INIT
10000000	APR_MNT_RunSc	Call APR_MNT.Ru	1 - User	☑	1 - Scheduled	1 - Success	1/1/2017 11:00:00 P	1/1/2017 11:00:00 P	1/1/2017 11:00:00 PM	System	9/11/2014 01:2	INIT
10000000	APR_MPI_Populat	APR_MPI_Populat	1 - User	☐	0 - New	1 - Success	2/5/2016 12:00:00 P	2/5/2016 12:00:00 P	2/5/2016 12:40:56 PM	System	9/1/2011 11:50	INIT
10000001	DELETE_FULL_JO	Delete full Job His	1 - User	☑	1 - Scheduled	1 - Success	1/2/2017 11:00:10	1/2/2017 11:00:12 A	1/2/2017 11:00:12 AM	System	1/25/2016 04:2	ADMIN
10000000	DELETE_KPI_CAL	Call Archiving Pro	1 - User	☑	1 - Scheduled	1 - Success	1/2/2017 11:00:00	1/2/2017 11:00:02 A	1/2/2017 11:00:02 AM	System	1/25/2016 04:2	ADMIN
10000001	DELETE_TRANSA	Delete Transactio	1 - User	☑	1 - Scheduled	1 - Success	1/2/2017 11:00:03	1/2/2017 11:00:09 A	1/2/2017 11:00:09 AM	System	1/25/2016 04:2	ADMIN
10000001	KPI_APR_Availabi	Equipment Availa	1 - User	☑	1 - Scheduled	1 - Success	1/2/2017 11:40:00	1/2/2017 11:40:00 A	1/2/2017 11:40:00 AM	System	7/1/2010 10:09	ADMIN
10000001	KPI_APR_Electrici	Electricity Consu	1 - User	☑	1 - Scheduled	1 - Success	1/2/2017 11:45:00	1/2/2017 11:45:00 A	1/2/2017 11:45:00 AM	System	1/25/2011 09:1	Tom

Jobs Currently Executed

The list of jobs currently executed can be found on the Jobs Currently Executed screen accessible from Job Scheduler main screen:

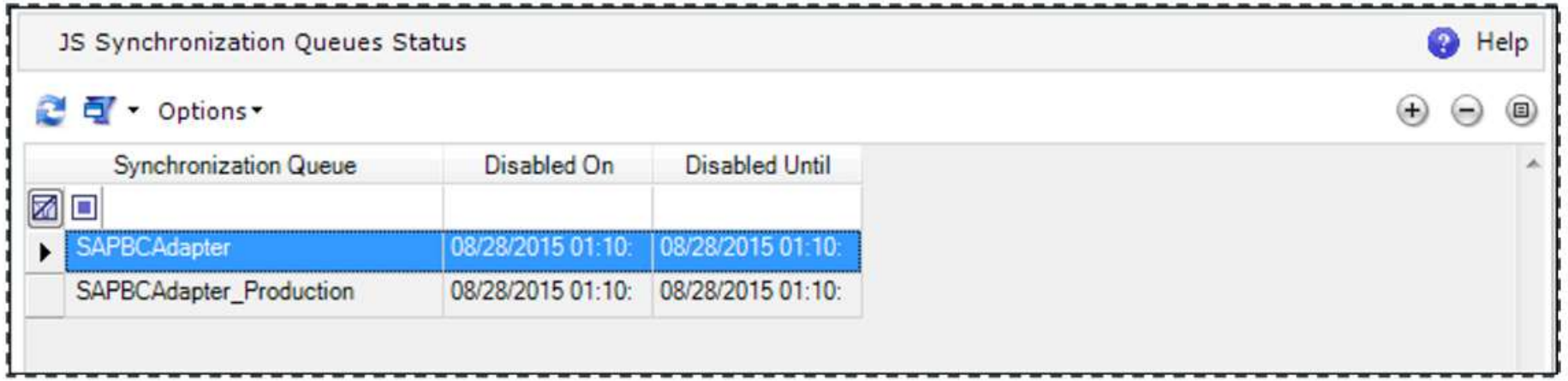
- Used by JS
- Displays jobs that are currently being executed



Synchronization Queues Status

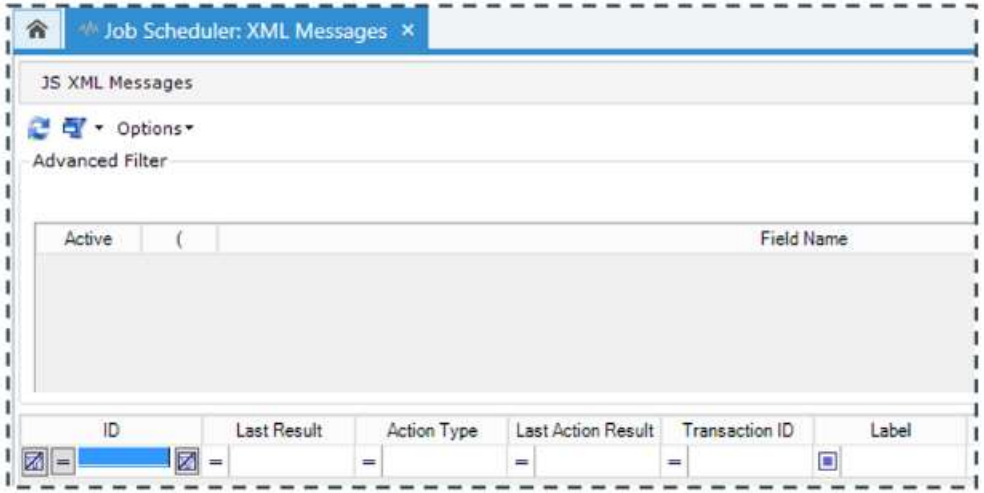
The Synchronization Queue Status grid displays all the disabled queues (and only the disabled ones) together with the Disabled On and Disabled Until dates.

- Used by JS
- Allows user to manually disable a synchronization queue
- Can specify start and stop times for disabling the queue
- When disabled, pauses the processing of all messages assigned to the queue



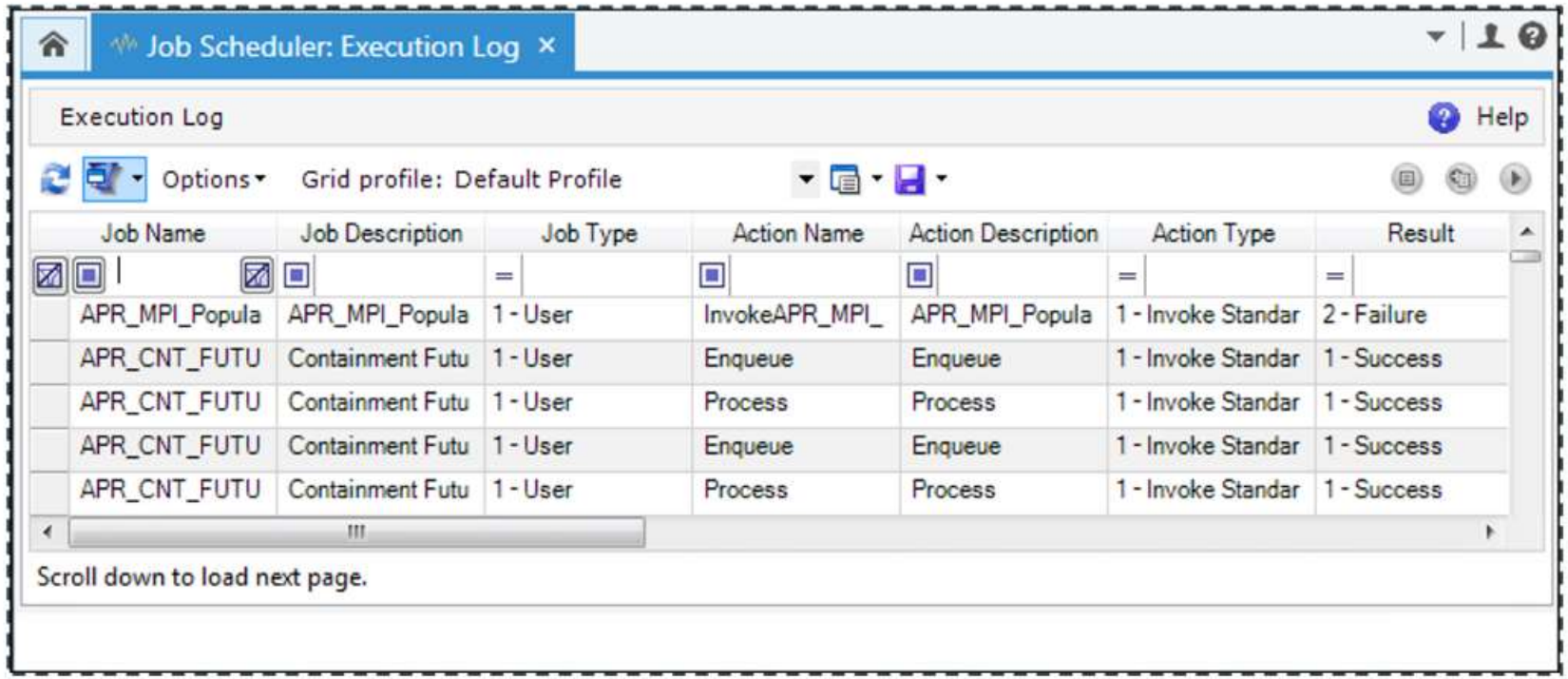
XML Messages Screen

- Displays all jobs related to XML messages originating from BI
- Mainly used for backward compatibility
- Displays the history of executed jobs
- Allows re-executing of jobs
- Allows viewing of job details



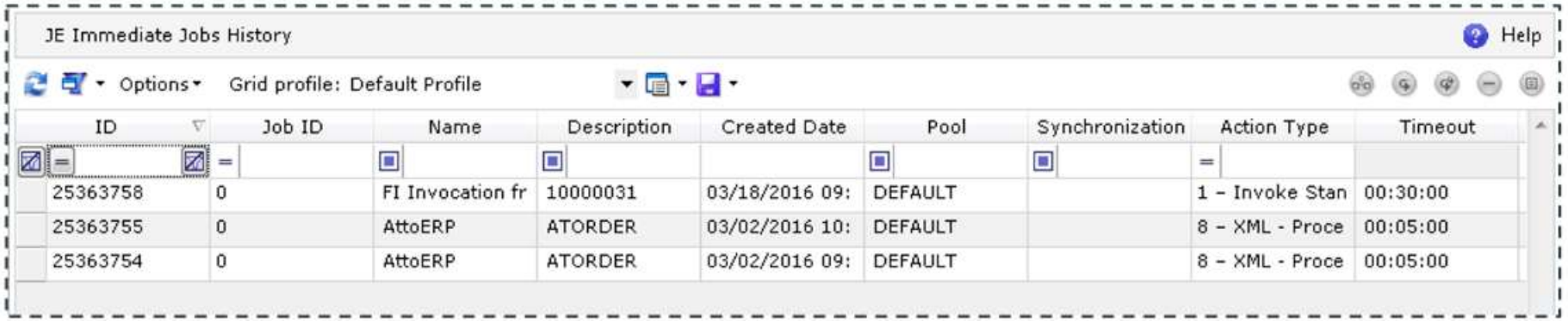
Execution Log

- Used by JS
- Displays history of executed job actions
- Allows re-executing job actions
- Allows viewing job action details



Immediate Jobs History

- Used by JE
- Displays JE History
- Allows viewing of failed and processed jobs
- Commonly used to view failed integration messages
- Allows viewing and editing of detailed job properties
- Allows two ways to reprocess jobs:
  - Reprocess
  - Edit and reprocess

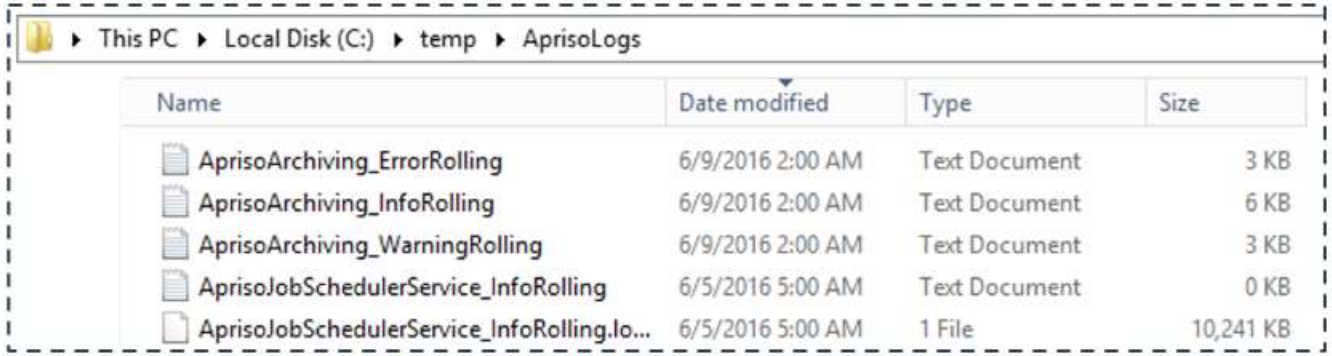


ID	Job ID	Name	Description	Created Date	Pool	Synchronization	Action Type	Timeout
25363758	0	FI Invocation fr	10000031	03/18/2016 09:	DEFAULT		1 - Invoke Stan	00:30:00
25363755	0	AttoERP	ATORDER	03/02/2016 10:	DEFAULT		8 - XML - Proce	00:05:00
25363754	0	AttoERP	ATORDER	03/02/2016 09:	DEFAULT		8 - XML - Proce	00:05:00

Logging

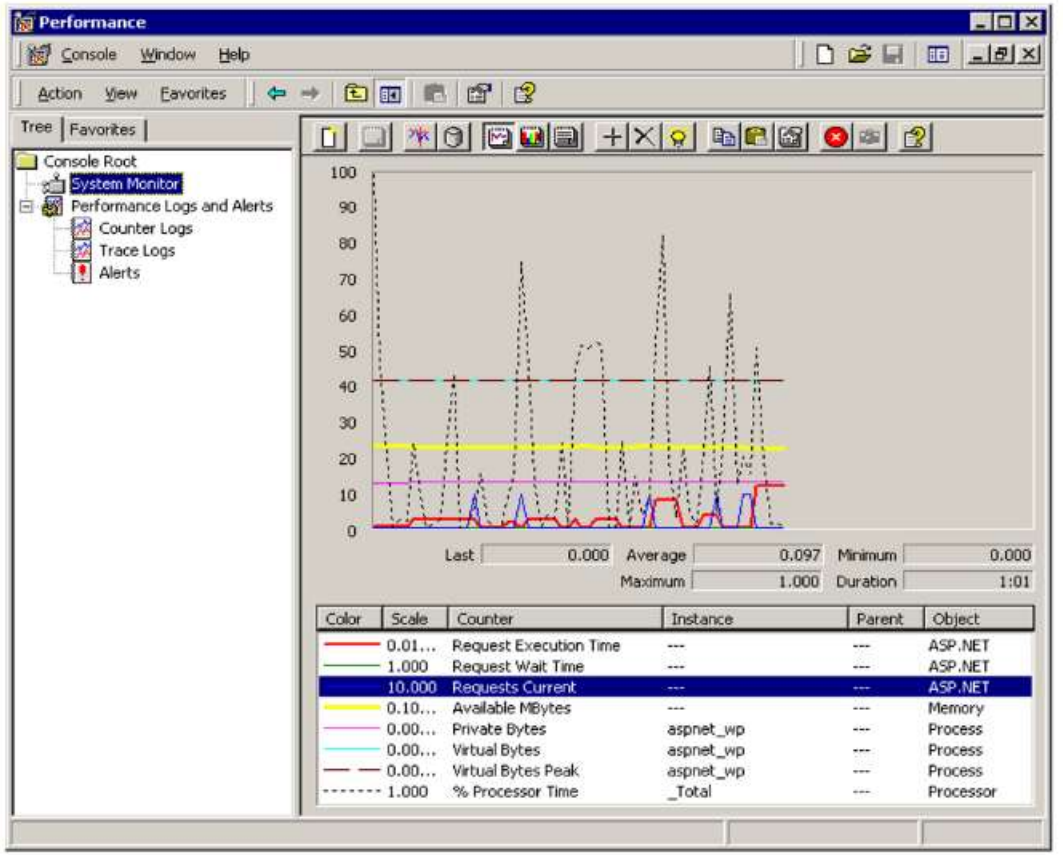
As with every other application within DELMIA Apriso, the Job Executor and Job Scheduler keep track of all activities in execution logs. By default these logs can be found in the C:\Temp\AprisoLogs\directory.

- JE and JS log all activity using the DELMIA Apriso logging framework
- The C:\Temp\AprisoLogs\ directory contains the following log files:
  - AprisoJobExecutorService_*.log for JE
  - AprisoSchedulingService_*.log for JS
- Commonly used to find more detailed error messages for failed jobs
- Performance counters can be used to observe the health of the system



Performance Counters

- DELMIA Apriso offers a choice of several performance counters that can be used in the Windows Performance Monitor to observe the performance of JE
- These counters can help users identify bottlenecks, reduce inefficiency, and boost productivity
- Examples of performance counter values for Background Processing functionality are published by DELMIA Apriso:
  - Scheduler Active Executor Threads
  - Scheduler Job Execution Duration
  - Scheduler Job Executions
  - Scheduler Job Executions/sec
  - Scheduler Job Executions (successful)
  - Scheduler new jobs
  - Executor Job Execution Duration
  - Executor Successful Job Executions/sec
  - Executor new jobs
  - Executor Active Executor Threads



Troubleshooting

How do I know whether JE has stopped running or is just running slowly?

- Check logs - if there is a new activity then it is running. If it has stopped, then restart the JE service. If it is doing something then check the Immediate Jobs - a job may be blocking the synchronization queue if it is set to retry 5 times and wait 5 minutes between retries. This effectively blocks the queue for 25 minutes

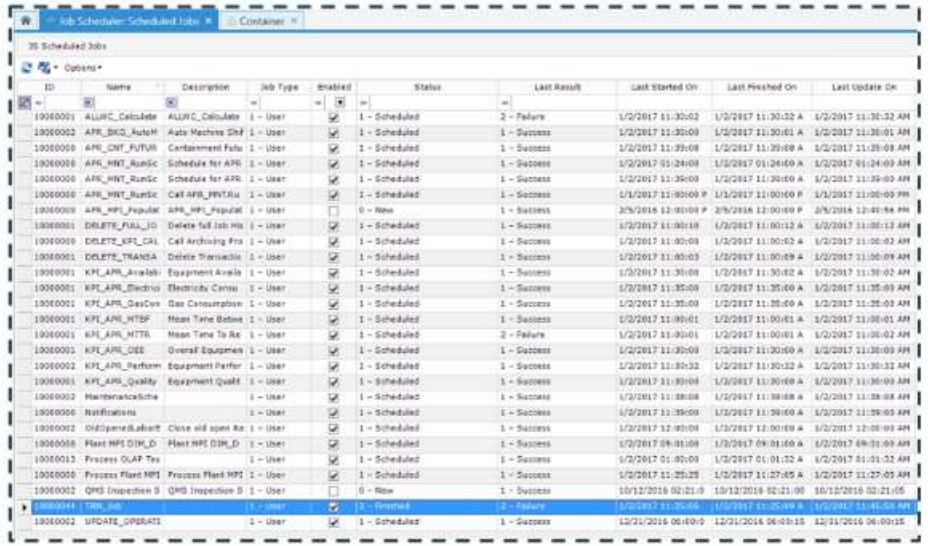
How do I remove a job from the queue if its status is 'running' ?

- Stop the JE service and then remove the job. Do it either via the Immediate Jobs screen or directly in the database. It has to be agreed with the client so that they resubmit the job again

If I have multiple JE instances running on my server (not very frequent), how do I know which one has the job that is causing trouble?

- The Job Pools screen shows the relationship between Job Pools and JE instances (installing additional JE instances is rare and requires DELMIA Apriso Support help)

LAB 1: Create a User Defined Job



Task:

- Create a job
- Add a job action
- Run the job
- Schedule the job

What you will learn:

- How to create a job,
- How to run it manually
- How to schedule it to run on a re-occurring basis

Training environment:

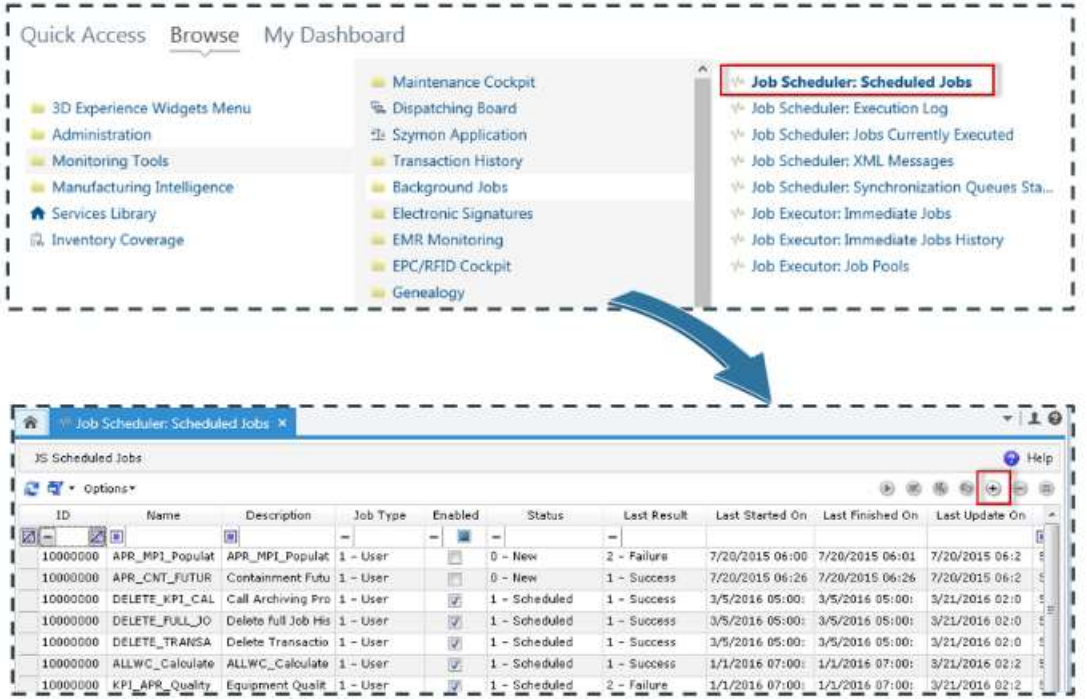
- In case of any technical problems, please contact DELMIA.Apriso.training@3ds.com

Remember to use the following to login and name Screens thorough this entire training:
- TRN<yourinitials> if your are an external self-paced learner
- TRN<yourtrigram> if you are a 3DS employee self-paced learner

20 min

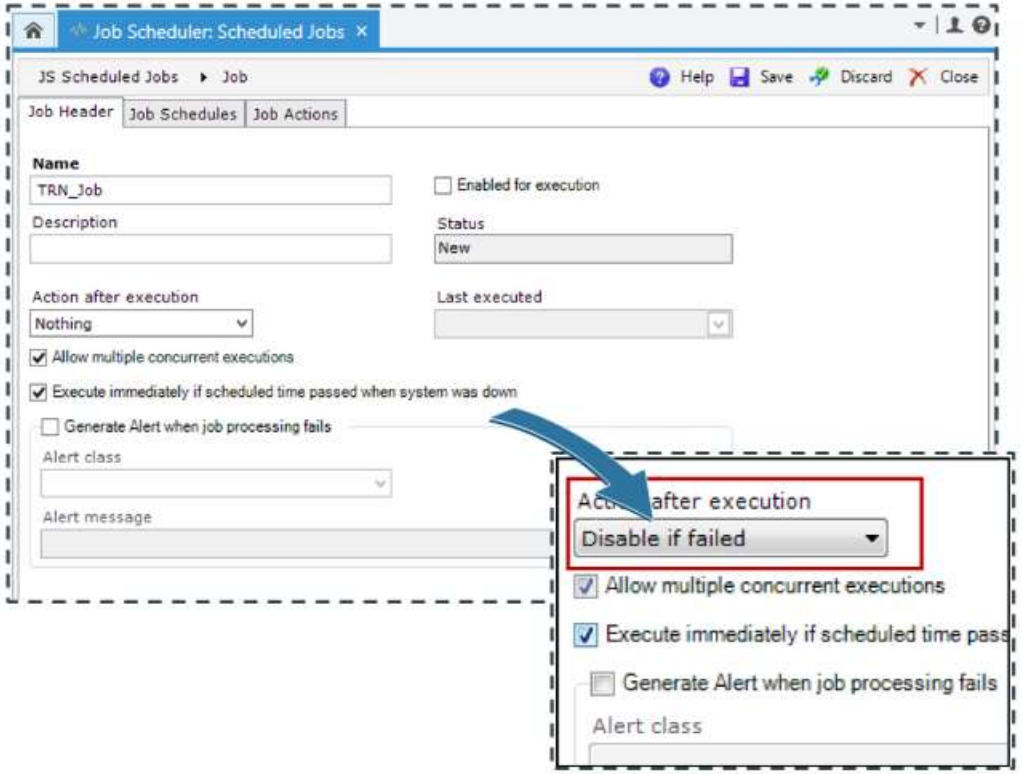
LAB 1: Create TRNXX_Job

- Go to Desktop Client and open the Job Scheduler: **Scheduled Jobs M&M Screen**
- Create a new job called **TRNXX_Job**



LAB 1: Set Action After Execution

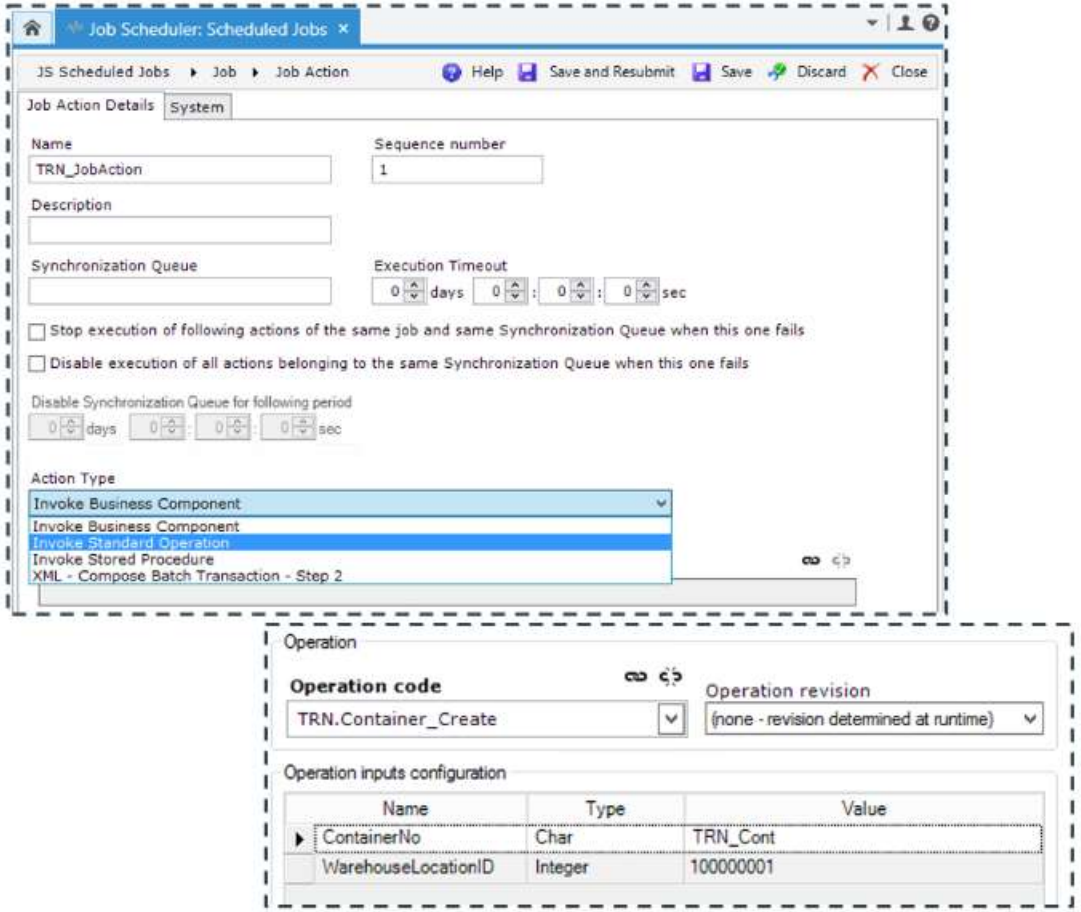
- Set the Action after Execution setting to **Disable if failed**
- Save your changes, but do not close this window



LAB 1: Create TRNXX_JobAction

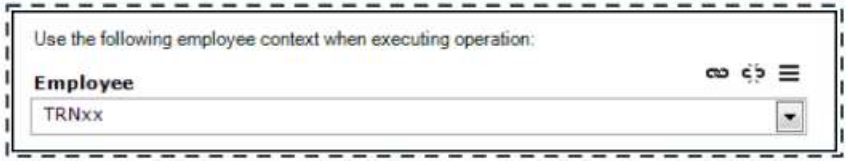
- Select the Job Actions tab
- Create a new Job Action
- Name your Job Action **TRNXX_JobAction**
- Set the Action Type to **Invoke Standard Operation**
- Assign the Standard Operation **TRN.Container_Create** to the Operation Code
- Add the Value TRNXX_Cont to the Input Name of ContainerNo
- Add the **Value 100000001** to the Input Name of WarehouseLocationID

This Operation will create a new Container named TRNXX_Cont.



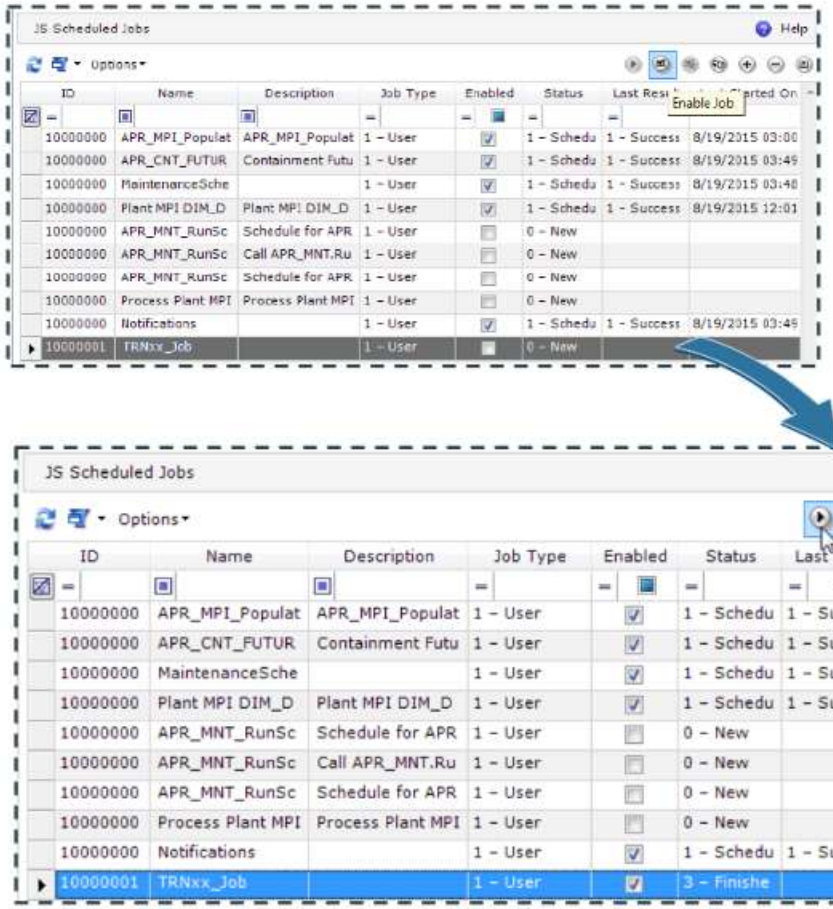
LAB 1: Assign the Employee

- Assign the **Employee Admin** (yourself)
- Save the changes
- Close both, the Job Action Details screen and the Job Actions screen so that you are back to the list of all Jobs in the system



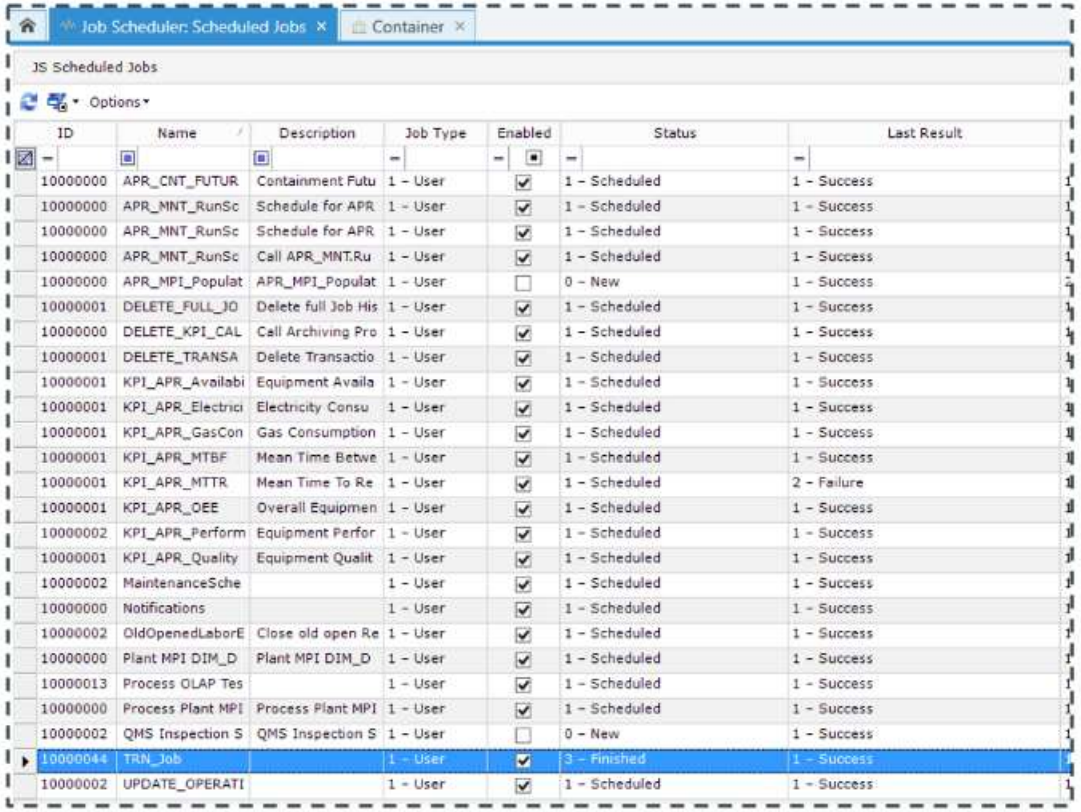
LAB 1: Run Your Job

- Find your job in the list of Jobs and select it
- Enable your job
- Run your job



LAB 1: Check Status of the Job

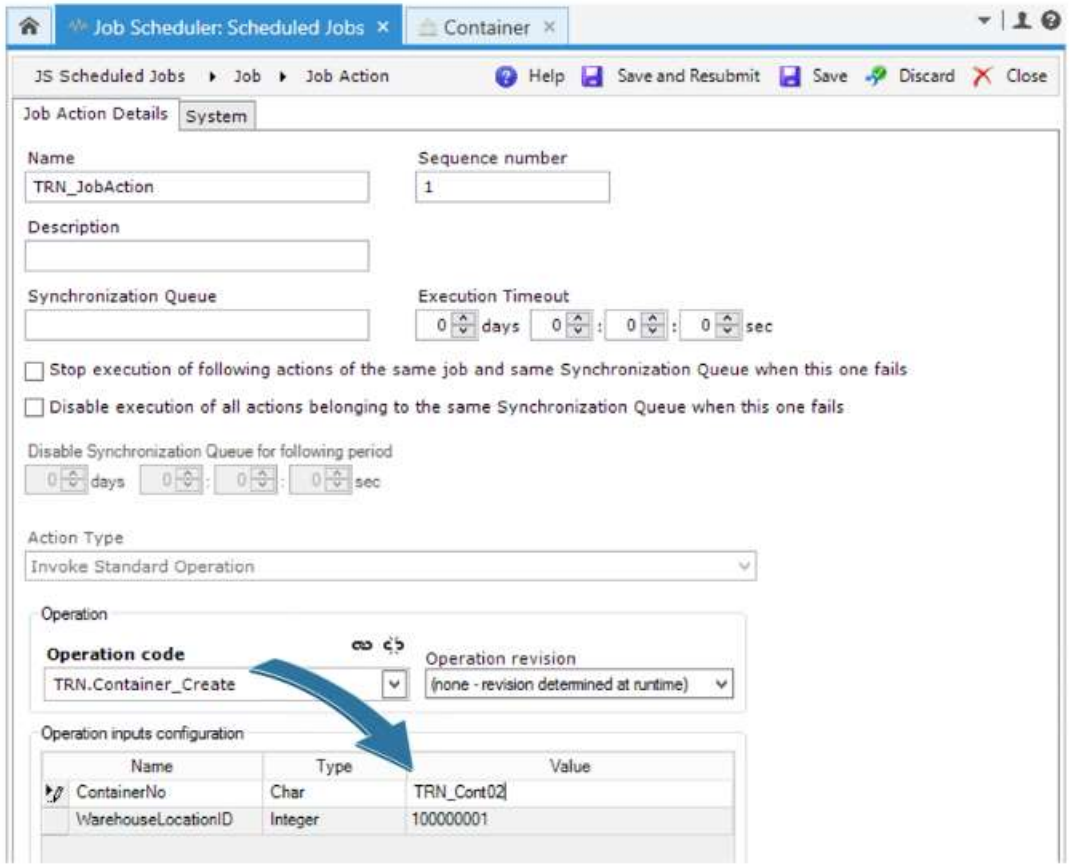
- Confirm that your job has run by checking the status of the Job. The Last Result should be Success
- Open **the Container M&M** screen and check to see if your container has been created
  - Path to Screen: Administration -> Data Administration -> Container Maintenance -> Container



ID	Name	Description	Job Type	Enabled	Status	Last Result
10000000	APR_CNT_FUTUR	Containment Futu	1 - User	☑	1 - Scheduled	1 - Success
10000000	APR_MNT_RunSc	Schedule for APR	1 - User	☑	1 - Scheduled	1 - Success
10000000	APR_MNT_RunSc	Schedule for APR	1 - User	☑	1 - Scheduled	1 - Success
10000000	APR_MNT_RunSc	Call APR_MNT.Ru	1 - User	☑	1 - Scheduled	1 - Success
10000000	APR_MPI_Populat	APR_MPI_Populat	1 - User	☐	0 - New	1 - Success
10000001	DELETE_FULL_JO	Delete full Job His	1 - User	☑	1 - Scheduled	1 - Success
10000000	DELETE_KPI_CAL	Call Archiving Pro	1 - User	☑	1 - Scheduled	1 - Success
10000001	DELETE_TRANSA	Delete Transactio	1 - User	☑	1 - Scheduled	1 - Success
10000001	KPI_APR_Availabi	Equipment Availa	1 - User	☑	1 - Scheduled	1 - Success
10000001	KPI_APR_Electrici	Electricity Consu	1 - User	☑	1 - Scheduled	1 - Success
10000001	KPI_APR_GasCon	Gas Consumption	1 - User	☑	1 - Scheduled	1 - Success
10000001	KPI_APR_MTBF	Mean Time Betwe	1 - User	☑	1 - Scheduled	1 - Success
10000001	KPI_APR_MTTR	Mean Time To Re	1 - User	☑	1 - Scheduled	2 - Failure
10000001	KPI_APR_OEE	Overall Equipmen	1 - User	☑	1 - Scheduled	1 - Success
10000002	KPI_APR_Perform	Equipment Perfor	1 - User	☑	1 - Scheduled	1 - Success
10000001	KPI_APR_Quality	Equipment Qualit	1 - User	☑	1 - Scheduled	1 - Success
10000002	MaintenanceSche		1 - User	☑	1 - Scheduled	1 - Success
10000000	Notifications		1 - User	☑	1 - Scheduled	1 - Success
10000002	OldOpenedLaborE	Close old open Re	1 - User	☑	1 - Scheduled	1 - Success
10000000	Plant MPI DIM_D	Plant MPI DIM_D	1 - User	☑	1 - Scheduled	1 - Success
10000013	Process OLAP Tes		1 - User	☑	1 - Scheduled	1 - Success
10000000	Process Plant MPI	Process Plant MPI	1 - User	☑	1 - Scheduled	1 - Success
10000002	QMS Inspection S	QMS Inspection S	1 - User	☐	0 - New	1 - Success
10000044	TRN_Job		1 - User	☑	3 - Finished	1 - Success
10000002	UPDATE_OPERATI		1 - User	☑	1 - Scheduled	1 - Success

LAB 1: Change Container Number

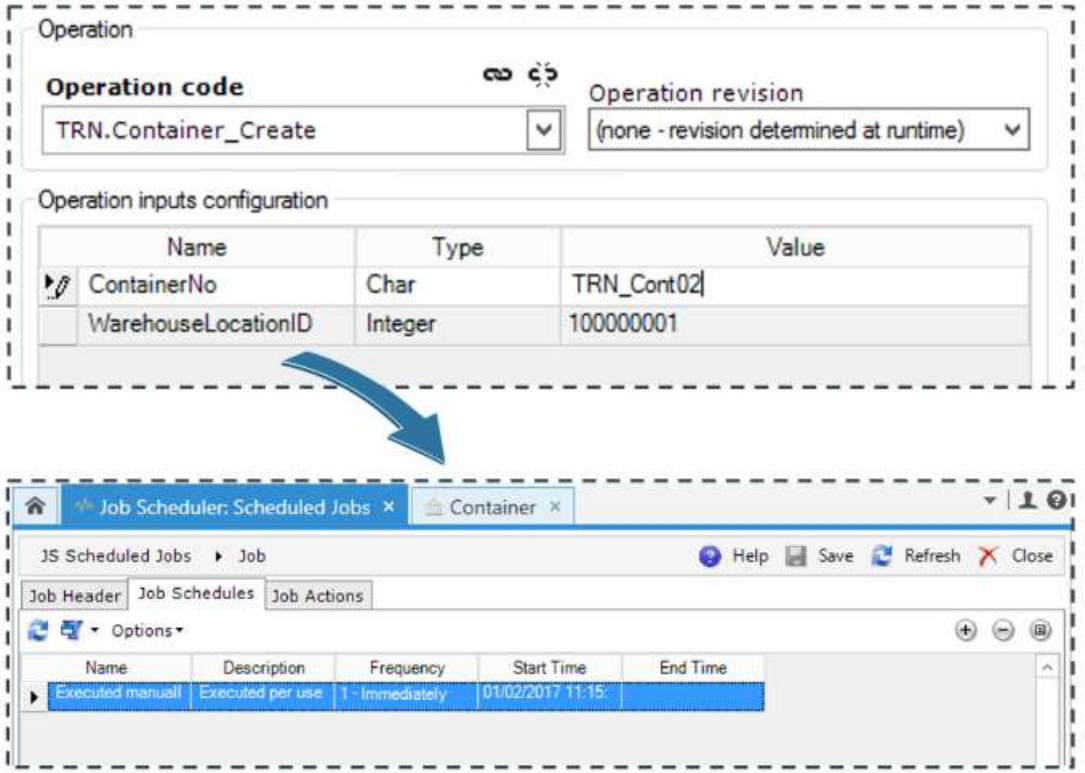
- Return to the Job Scheduler M&M screen and open your Job
- Change your Container Number in the Job Actions from TRNXX_Cont to TRNXX_Cont02
- Save and Close your Job Action



LAB 1: Check Job Schedules

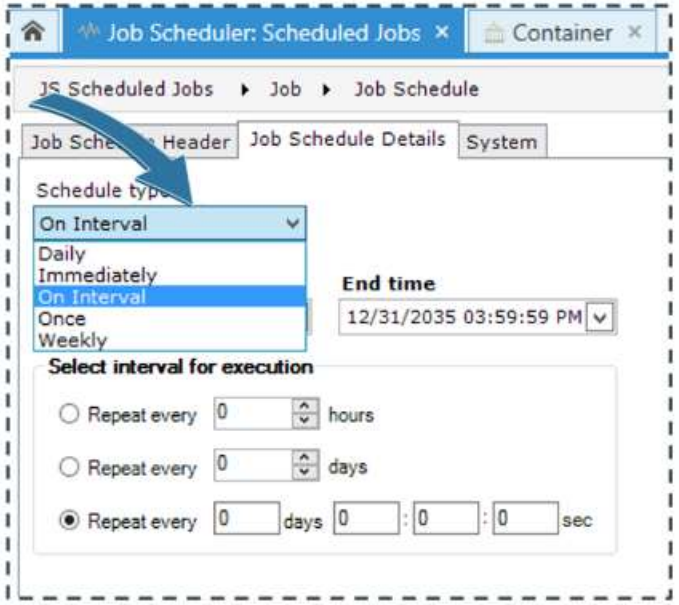
- Open the Job Schedules tab

You will notice that you have a record here that shows one execution. This was created when you manually ran the job.



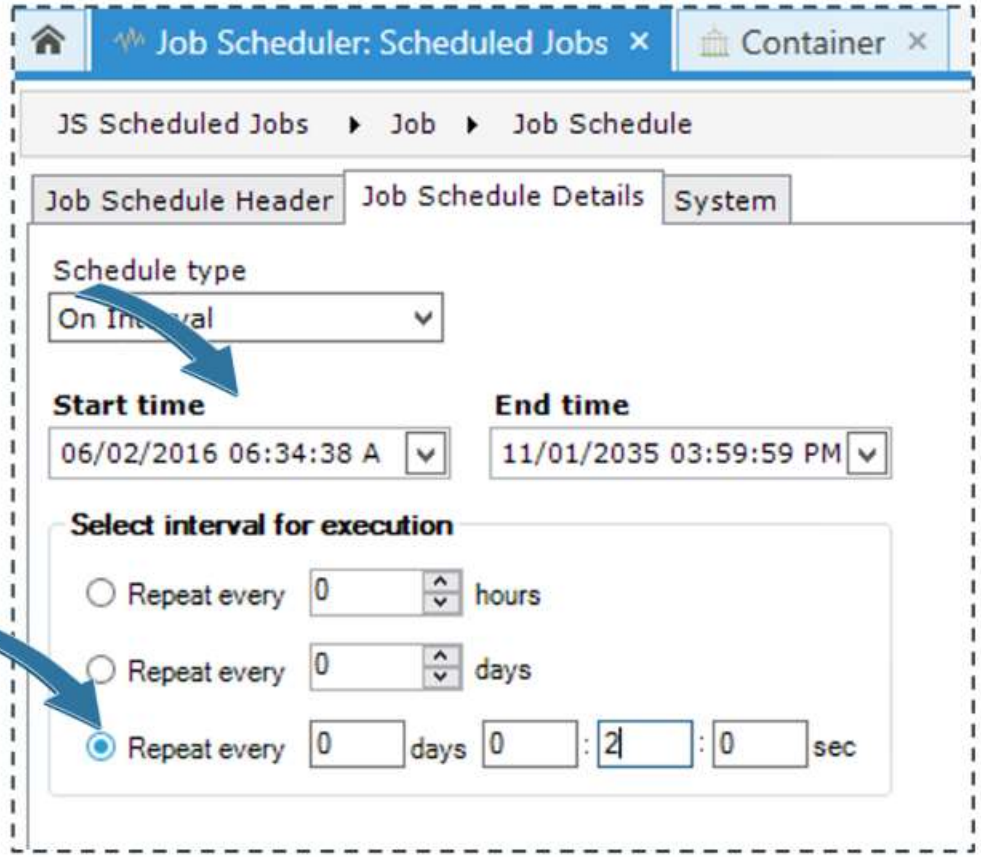
LAB 1: Add TRNXX_JobSchedule

- Add a new Schedule by clicking on icon.
- Set the name of your Schedule to **TRNXX_JobSchedule**
- Open the Job Schedule Details and from Schedule Type drop-down menu tab choose On Interval



LAB 1: Schedule Properties

- Assign the following values to the Schedule Properties:
  - Start Time: pick any time that is in the past
  - End Time: pick any time that is at least 1 hour in the future
  - Set the seconds for the Repeat Every interval to 2 minutes
- Save your changes and close the properties screen so that you are looking at your job in the main Job Scheduler grid



LAB 1: Check Results

- Refresh the screen until you see the Last Finished On field updated to the current time. This means the job has finished running
- Check to make sure your container was created (TRNXX_Cont02)

Now wait two minutes - this is the interval time scheduled for the job.

- Go back to the Job Scheduler screen and check your Job. If it has been more than 2 minutes since it last ran, your job should show a "Last Result" of "Failure" and it should be in a "Disabled" state. This is because the Job is calling a standard operation that is trying to create a container in the database that already exists (we created it the first time the job ran)