ETL JOB Management - Developer’s Guide

Sita Sathiraju

September 8, 2020

# Table of Contents

[Table of Contents 2](#_Toc50497351)

[1 Overview 4](#_Toc50497352)

[1.1 Data Models 5](#_Toc50497353)

[1.1.1 ETL Job Management – Data Model 5](#_Toc50497354)

[1.2 ETL Job Processing 6](#_Toc50497355)

[1.2.1 Table(s) 6](#_Toc50497356)

[1.2.2 View(s) 6](#_Toc50497357)

[1.2.3 Sequence(s) 6](#_Toc50497358)

[1.2.4 Trigger(s) 6](#_Toc50497359)

[1.2.5 Package(s) 7](#_Toc50497360)

[1.2.6 Linux Script File(s) 7](#_Toc50497361)

[1.2.7 Linux Property File(s) 7](#_Toc50497362)

[1.2.8 Linux SQL File(s) 7](#_Toc50497363)

[1.2.9 Linux Java File(s) 8](#_Toc50497364)

[2 Development 8](#_Toc50497365)

[2.1 etl\_job.properties file 8](#_Toc50497366)

[2.2 .set\_env file 8](#_Toc50497367)

[2.3 etl\_job\_insert\_script.sql 10](#_Toc50497368)

[2.4 Project script files 12](#_Toc50497369)

[3 Install 13](#_Toc50497370)

[3.1 Pre-requisites 13](#_Toc50497371)

[3.2 Oracle Database 13](#_Toc50497372)

[3.3 Linux 14](#_Toc50497373)

[3.4 Crontab 15](#_Toc50497374)

[4 Test 16](#_Toc50497375)

[4.1.1 Through Crontab 16](#_Toc50497376)

[4.1.2 Adhoc 16](#_Toc50497377)

[5 Administration 16](#_Toc50497378)

[5.1 Global Configuration 16](#_Toc50497379)

[5.2 Configuring Jobs 17](#_Toc50497380)

[5.3 ETL\_JOB Package 17](#_Toc50497381)

[5.3.1 Procedure ENABLE\_ETL\_JOB 17](#_Toc50497382)

[5.3.2 Procedure DISABLE\_ETL\_JOB 18](#_Toc50497383)

[5.3.3 Procedure COMPLETE\_ETL\_JOB 19](#_Toc50497384)

[5.3.4 Procedure RESET\_ETL\_JOB 20](#_Toc50497385)

[5.3.5 Procedure CANCEL\_ETL\_JOB 21](#_Toc50497386)

[5.3.6 Procedure PURGE\_ETL\_JOB\_DATA 22](#_Toc50497387)

[6 Glossary 22](#_Toc50497388)

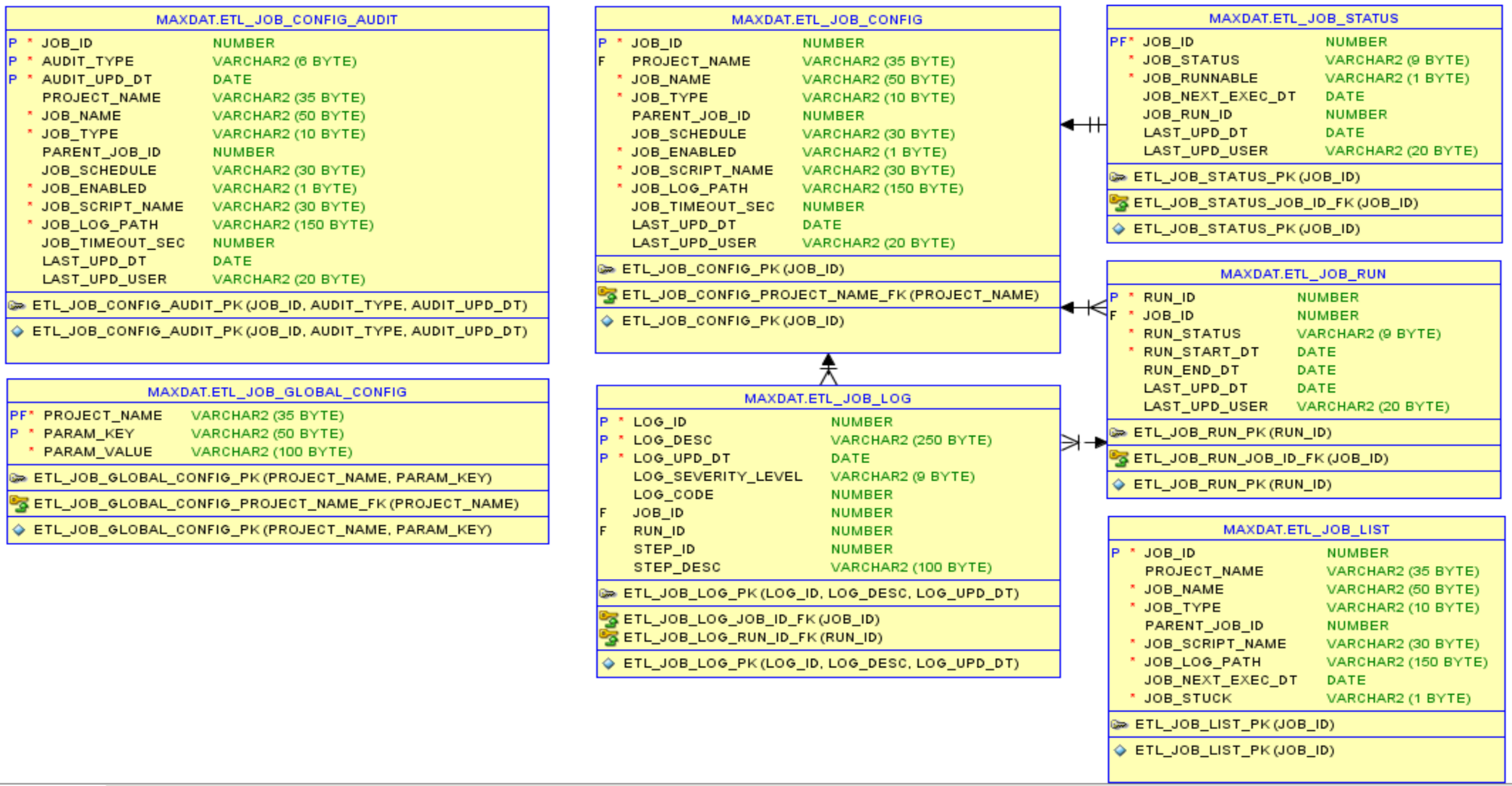
# Overview

This is a developer’s guide for writing code to populate the ETL Job Management Process. It consists of an overview of how things work, a development section describing the implementation steps and how to test and deploy.

## Data Models

### ETL Job Management – Data Model

The ETL Job data model is a normalized data model containing a record lookup values of the jobs schedule. The data model design source for the diagram can be found in SVN: maxdat\BPM\ETL\_Job\_Control\datamodel\

****

## ETL Job Processing

Linux scripts call various procedures of ETL\_JOB package to identify the jobs to be executed, add a new run instance for each job, sets the status to complete once execution is complete. All the job configuration, status, run details, logs, etc., will be stored in the Oracle database

### Table(s)

* ETL\_JOB\_CONFIG : Configuration table for Jobs
* ETL\_JOB\_CONFIG\_AUDIT : Audit table that tracks all changes to Configuration table
* ETL\_JOB\_STATUS : Current status for each job, next execution time, etc.,
* ETL\_JOB\_RUN : History table for all job runs
* ETL\_JOB\_LIST : Staging table that gets refreshed by ETL control job
* ETL\_JOB\_GLOBAL\_CONFIG : Global parameters for ETL job control process
* ETL\_JOB\_LOG : Hold all necessary logs and error messages

### View(s)

* ETL\_JOB\_CONFIG\_AUDIT\_SV : View for ETL\_JOB\_CONFIG\_AUDIT table
* ETL\_JOB\_STATUS\_SV : View for ETL\_JOB\_STATUS table
* ETL\_JOB\_LIST\_SV : View for ETL\_JOB\_LIST table
* ETL\_JOB\_GLOBAL\_CONFIG\_SV : View for ETL\_JOB\_GLOBAL\_CONFIG table
* ETL\_JOB\_LOG\_SV : View for ETL\_JOB\_LOG table
* ETL\_JOB\_RUN\_SV : View for ETL\_JOB\_RUN table
* ETL\_JOB\_CONFIG\_SV : View for ETL\_JOB\_CONFIG table

### Sequence(s)

* SEQ\_ETL\_JOB\_CONFIG\_ID : Sequence generator for ETL\_JOB\_CONFIG.JOB\_ID
* SEQ\_ETL\_JOB\_RUN\_ID : Sequence generator for ETL\_JOB\_RUN.RUN\_ID
* SEQ\_ETL\_LOG\_ID : Sequence generator for ETL\_JOB\_LOG.LOG\_ID

### Trigger(s)

* AUD\_TR\_ETL\_JOB\_CONFIG : Tracks all Update/Delete changes to ETL\_JOB\_CONFIG table in ETL\_JOB\_CONFIG\_AUDIT table

### Package(s)

#### ETL\_JOB

* Procedure SET\_ETL\_JOBS : Identifies list of jobs to execute/ got stuck
* Procedure ADD\_ETL\_JOB : starts a new run instance of the job
* Procedure ENABLE\_ETL\_JOB : Enables any job, including/excluding dependent jobs
* Procedure DISABLE\_ETL\_JOB : Disables any job, including/excluding dependent jobs
* Procedure COMPLETE\_ETL\_JOB : Completes the existing run of any job as well as parent jobs
* Procedure RESET\_ETL\_JOB : Resets the existing run of any job, including/excluding dependent jobs
* Procedure CANCEL\_ETL\_JOB : Cancels the existing run of any job, including/excluding dependent jobs
* Procedure ADD\_ETL\_LOG : inserts log entry into ETL\_JOB\_LOG table
* Procedure PURGE\_ETL\_JOB\_DATA : Purges data from ETL\_JOB\_CONFIG\_AUDIT, ETL\_JOB\_RUN, and ETL\_JOB\_LOG tables based on global

parameters defined in ETL\_JOB\_GLOBAL\_CONFIG table

* Function GET\_NEXT\_JOB\_SCHEDULE : get the next job schedule date/time
* Function IS\_NUMBER\_VALID : sub function for GET\_NEXT\_JOB\_SCHEDULE
* Function PARSE\_NUMBER\_RANGE : sub function for GET\_NEXT\_JOB\_SCHEDULE
* Function GET\_NEXT\_NUMBER : sub function for GET\_NEXT\_JOB\_SCHEDULE
* Function GET\_MIN\_NUMBER : sub function for GET\_NEXT\_JOB\_SCHEDULE

### Linux Script File(s)

* etl\_job\_control.bash : main ETL job control shell script that gets configured in crontab to run every 5 minutes
* etl\_job\_exec.bash : shell script that gets executed for each job
* decr.sh : shell script decrypts the DB\_MAXDAT\_PASSWORD in kettle\_properties file by calling

KettleDecryptPassword.class

### Linux Property File(s)

* etl\_job.properties : this properly file is used to locate .set\_env file for the project

### Linux SQL File(s)

* set\_etl\_jobs.sql : executes SET\_ETL\_JOBS procedure of ETL\_JOB package
* add\_etl\_job.sql : executes ADD\_ETL\_JOB procedure of ETL\_JOB package
* complete\_etl\_job.sql : executes COMPLETE\_ETL\_JOB procedure of ETL\_JOB package
* add\_etl\_log.sql : executes ADD\_ETL\_LOG procedure of ETL\_JOB package
* get\_etl\_jobs.sql : SQL to extract list of jobs to execute from ETL\_JOB\_LIST table
* get\_etl\_stuck\_jobs : SQL to extract list of stuck jobs from ETL\_JOB\_LIST table

### Linux Java File(s)

* KettleDecryptPassword.class : this class will call Kettle java functions to decrypt DB\_MAXDAT\_PASSWORD in kettle\_properties file

# Development

ETL job management is designed to work for any existing ETL projects. Any ETL project can be configured into ETL job management process by setting some configurations, modify the existing scripts by commenting any code related to file check. The below steps details how an ETL project can be updated for the ETL job management.

## etl\_job.properties file

This file should be modified to set the file name and location of .set\_env for the project. ETL job management process will read this file to locate .set\_env for the project, and load it prior to job execution. The process will also identify the kettle.properties file location based on .set\_env.

# Set the .set\_env file name and absolute path

SET\_ENV\_FILENAME=.set\_env

SET\_ENV\_FILEPATH=/u01/maximus/maxdat/IL8/config

#

Reference: This file is located in SVN:\maxdat\trunk\Kettle8\IL\scripts\ for ILEB project. So, a copy of this file can be copied over to project specific path and make necessary changes.

## .set\_env file

.set\_env file for the project should be modified to add the below entries.

* ETL\_JOBCONTROL\_SCRIPTS\_PATH is used by ETL Job management process to search all the scripts configured in ETL\_JOB\_CONFIG.JOB\_SCRIPT\_NAME. This variable can accept multiple folders that should be separated by colon “:”, without trailing “/” at the end for each path.
* The path “/u01/maximus/maxdat/$STCODE/” should updated based on the project specific path, where the existing “scripts”, “config”, etc., folders are available for the project
* Oracle variables path should be updated based on the sqlplus installation.

#

# ---- ETL JOB CONTROL PROCESS VARIABLES ----

export ETL\_JOBCONTROL\_SCRIPTS\_PATH="/u01/maximus/maxdat/$STCODE/scripts:/u01/maximus/maxdat/$STCODE/scripts/ManageWork"

export ETL\_JOBCONTROL\_LOG\_PATH="/u01/maximus/maxdat/$STCODE/logs"

export ETL\_JOBCONTROL\_DATA\_PATH="/u01/maximus/maxdat/$STCODE/data"

export ETL\_JOBCONTROL\_SQL\_PATH="/u01/maximus/maxdat/$STCODE/sql"

export KETTLE\_PROPERTIES\_PATH="$KETTLE\_HOME/.kettle"

#

# ---- ETL JOB CONTROL PROCESS VARIABLES : ORACLE RELATED ----

export ORACLE\_HOME=/u01/app/appadmin/product/oracle/instantclient

export LD\_LIBRARY\_PATH=${LD\_LIBRARY\_PATH}:/u01/app/appadmin/product/oracle/instantclient

export DB\_MAXDAT\_CLIENT=/u01/app/appadmin/product/oracle/instantclient/sqlplus

#

Reference: This file is located in SVN:\maxdat\trunk\Kettle8\IL\config\ for ILEB project. .set\_env for the specific project should be updated with the above changes and re-deployed into production as part of the ETL job management process.

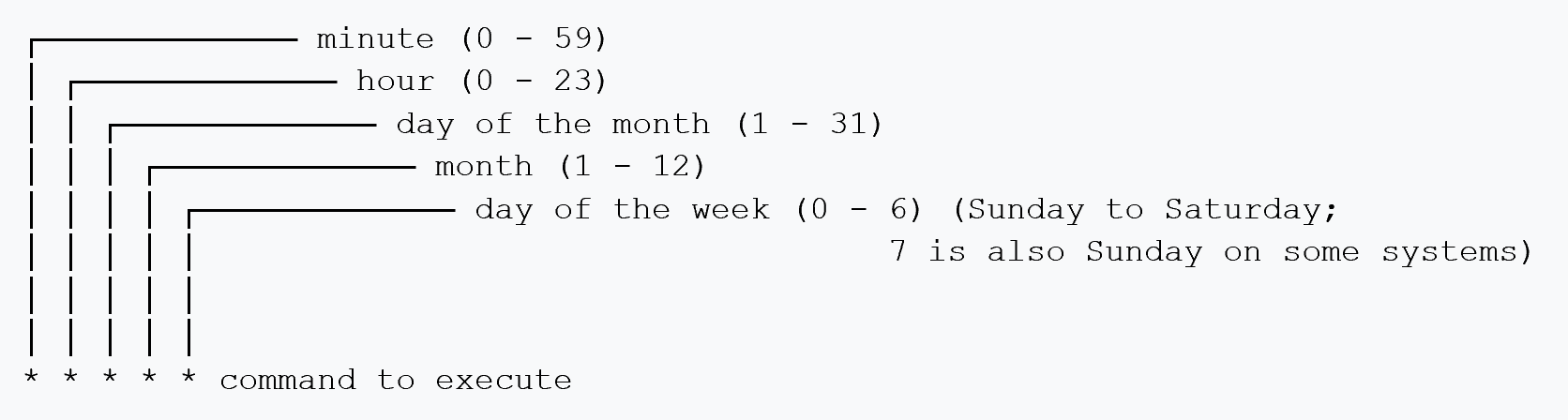
## etl\_job\_insert\_script.sql

This script should be modified to insert data into ETL\_JOB\_GLOBAL\_CONFIG and ETL\_JOB\_CONFIG tables.

* DML for ETL\_JOB\_GLOBAL\_CONFIG table:
  + All 5 entries are required for each project
  + PROJECT\_NAME should refer to MAXDAT.BPM\_PROJECT\_LKUP.NAME field
  + ETL\_AUDIT\_DATA\_RETENTION\_DAYS should be set to number of days the data in ETL\_JOB\_CONFIG\_AUDIT table should be retained
  + ETL\_RUN\_DATA\_RETENTION\_DAYS should be set to number of days the data in ETL\_JOB\_RUN table should be retained
  + ETL\_LOG\_DATA\_RETENTION\_DAYS should be set to number of days the data in ETL\_JOB\_LOG table should be retained
  + MAX\_PARALLEL\_JOBS\_ALLOWED should be set of maximum number of scripts from ETL\_JOB\_CONFIG that ETL job management process can execute in parallel. If there are more than this many jobs, those jobs will be skipped and picked up in the next run of ETL Job management process
  + JOB\_TIMEOUT\_SEC is the timeout for any job at project level. Any script specific timeout can be set at ETL\_JOB\_CONFIG.JOB\_TIMEOUT\_SEC field. ETL job management will check the current running jobs execution time and compares with these fields (if available), and raise a warning message “job is either stuck or long running…” if the execution time exceeds these fields in seconds.
* DML for ETL\_JOB\_ CONFIG table:
  + One entry for each script to be executed through ETL job management process
  + Here is the table structure and field descriptions:

|  |  |
| --- | --- |
| FIELD | SIGNIFICANCE |
| JOB\_ID | Unique ID for each job - auto generated using SEQ\_ETL\_JOB\_CONFIG\_ID sequence |
| PROJECT\_NAME | Foreign key to valid project names in BPM\_PROJECT\_LOOKUP.NAME |
| JOB\_NAME | Job Name - This help identify the job like: pa\_run\_bpm |
| JOB\_TYPE | Valid Values: PARENT, CHILD, INDIVIDUAL, ADHOC |
| PARENT\_JOB\_ID | In case of child Job, this gives the parent job id |
| JOB\_SCHEDULE | Defines the schedule for the job using CRONTAB format |
| JOB\_ENABLED | Valid Values: Y, N |
| JOB\_SCRIPT\_NAME | Actual script name WITHOUT script path, like : "pa\_run\_bpm.sh" |
| JOB\_LOG\_PATH | The path where log file will be created for the script. It should be a valid path without trailing “/” |
| JOB\_TIMEOUT\_SEC | Maximum execution time, after which a job is considered stuck - in seconds. |
| LAST\_UPD\_DT | Last Updated Datetime |
| LAST\_UPD\_USER | Last Updated User |

* + JOB\_SCHEDULE allows the same format as the CRON as given below. Custom code will be developed to accept all possible cron formats, but it may limit to fewer formats if the logic becomes more complex



* + JOB\_TYPE allows the below valid values based on the type of the job:

|  |  |
| --- | --- |
| INDIVIDUAL | No Parent or Child jobs. It should have its own JOB\_SCHEDULE |
| PARENT | It is expected to have some dependent/child jobs. It should have its own JOB\_SCHEDULE. PARENT job run is considered as complete only when all the child jobs are completed. |
| CHILD | Must have PARENT job and cannot have JOB\_SCHEDULE of its own. This gets triggered once PARENT job is complete. CHILD job can also have CHILD jobs further down the hierarchy. |
| ADHOC | Get disabled after each run. For additional runs, it should be enabled. If JOB\_SCHEDULE is provided, ADHOC job will be executed based on the next execution date/time based on the JOB\_SCHEDULE. If JOB\_SCHEDULE is not provided, it will picked up by the next ETL job control run. |

* + JOB\_SCRIPT\_NAME should be the actual shell script name without any path. The path will be read from the .set\_env.sh. ETL Job control process will check for this script in one of the script paths from the .set\_env.sh and executes the script if it exists in one of those paths.

Reference: This file is located in SVN:\maxdat\BPM\ILEB\createdb folder for ILEB project. So, a copy of this file can be copied over to project specific path and make necessary changes.

## Project script files

Any script files that are configured in ETL job management should be updated to comment out file check steps. ETL job management process has the in-built logic to hold the jobs from execution until the current instance is complete. So, any file based logic to check the job status and stop it, should be commented or removed from all the script files and should be re-deployed into production as part of the ETL job management process.

# Install

Please make necessary changes to any paths or project names provided below. The below instructions are from ILEB installation. Change them according to the project.

## Pre-requisites

* KETTLE 8.3 is installed

Verify if "pentaho\_8.3" folder exist in "/u01/app/appadmin/product/" folder

* kettle.properties file

This file is present in "…/.kettle/" folder and has the below properties updated.

DB\_MAXDAT\_NAME=<project specific>

DB\_MAXDAT\_HOSTNAME=<project specific>

DB\_MAXDAT\_PORT=<project specific>

DB\_MAXDAT\_USER=<project specific>

DB\_MAXDAT\_PASSWORD=encrypted <project specific>

DB\_MAXDAT\_SCHEMA=<project specific>

* SQL\*Plus instant client for Oracle 12c is installed

Verify if "sqlplus" file exist in "/u01/app/appadmin/product/oracle/instantclient" folder

## Oracle Database

* Install the below scripts:

svn://svn-staging.maximus.com/dev1d/maxdat/BPM/ETL\_Job\_Control/createdb/etl\_job\_control\_core.sql  
svn://svn-staging.maximus.com/dev1d/maxdat/BPM/ETL\_Job\_Control/createdb/etl\_job\_pkg.sql  
svn://svn-staging.maximus.com/dev1d/maxdat/BPM/ILEB/createdb/etl\_job\_insert\_script.sql

## Linux

* CREATE 2 NEW FOLDERS "data" and "sql" UNDER "/u01/maximus/maxdat/IL8/"

/u01/maximus/maxdat/IL8/data   
/u01/maximus/maxdat/IL8/sql

SET CHMOD ON THESE 2 FOLDERS TO "2755"

* CREATE A NEW FOLDER "ETLJobControl" UNDER "/u01/maximus/maxdat/IL8/logs/"

/u01/maximus/maxdat/IL8/logs/ETLJobControl

SET CHMOD ON THIS FOLDER TO "2755"

* COPY THE BELOW FILES FROM SVN INTO "/u01/maximus/maxdat/IL8/scripts/" folder

svn://svn-staging.maximus.com/dev1d/maxdat/trunk/ETL\_Job\_Control/ETL/scripts/etl\_job\_control.bash  
svn://svn-staging.maximus.com/dev1d/maxdat/trunk/ETL\_Job\_Control/ETL/scripts/etl\_job\_exec.bash  
svn://svn-staging.maximus.com/dev1d/maxdat/trunk/ETL\_Job\_Control/ETL/scripts/decr.sh  
svn://svn-staging.maximus.com/dev1d/maxdat/trunk/ETL\_Job\_Control/ETL/java/KettleDecryptPassword.class  
svn://svn-staging.maximus.com/dev1d/maxdat/trunk/Kettle8/IL/scripts/etl\_job.properties

SET CHMOD ON THESE 5 FILES TO "755" AND CONVERT THEM TO "UNIX" FORMAT

* COPY THE BELOW FILES (DO NOT RUN THEM) FROM SVN INTO "/u01/maximus/maxdat/IL8/sql/" folder

svn://svn-staging.maximus.com/dev1d/maxdat/trunk/ETL\_Job\_Control/ETL/sql/add\_etl\_job.sql  
svn://svn-staging.maximus.com/dev1d/maxdat/trunk/ETL\_Job\_Control/ETL/sql/add\_etl\_log.sql  
svn://svn-staging.maximus.com/dev1d/maxdat/trunk/ETL\_Job\_Control/ETL/sql/get\_etl\_jobs.sql  
svn://svn-staging.maximus.com/dev1d/maxdat/trunk/ETL\_Job\_Control/ETL/sql/get\_etl\_stuck\_jobs.sql  
svn://svn-staging.maximus.com/dev1d/maxdat/trunk/ETL\_Job\_Control/ETL/sql/set\_etl\_jobs.sql  
svn://svn-staging.maximus.com/dev1d/maxdat/trunk/ETL\_Job\_Control/ETL/sql/complete\_etl\_job.sql

SET CHMOD ON THESE 6 FILES TO "755" AND CONVERT THEM TO "UNIX" FORMAT

* SAVE A COPY OF THE SCRIPTS FILES ON THE SERVER, IN ORDER TO PLACE THEM BACK IN CASE OF ROLLBACK

Note: These are the list of files that were modified by commenting/removing the check file functionality that need to be re-deployed into PRD

/u01/maximus/maxdat/IL8/scripts/il\_run\_bpm.sh  
/u01/maximus/maxdat/IL8/scripts/il\_run\_emrs.sh

THE BELOW OBJECTS FROM SVN SHOULD BE COPIED OVER (REPLACE THE CURRENT FILES) TO "/u01/maximus/maxdat/IL8/scripts/" folder

svn://svn-staging.maximus.com/dev1d/maxdat/trunk/Kettle8/IL/scripts/il\_run\_bpm.sh  
svn://svn-staging.maximus.com/dev1d/maxdat/trunk/Kettle8/IL/scripts/il\_run\_emrs.sh

SET CHMOD ON THESE 2 FILES TO "755" AND CONVERT THEM TO "UNIX" FORMAT

* THE BELOW OBJECTS FROM SVN SHOULD BE COPIED OVER (REPLACE THE CURRENT FILES) TO "/u01/maximus/maxdat/IL8/config/" folder

svn://svn-staging.maximus.com/dev1d/maxdat/trunk/Kettle8/IL/config/.set\_env

## Crontab

* In this step, first give instruction to comment all the scripts that are being configured in the ETL job management and add the below lines to crontab
* Add below lines to Crontab:
* NOTE: The below crontab entries should have the escape character, backslash, prior to each %, i.e. $(date +\%Y-\%m-\%d). Jira is not showing the backslash character in the description or comment sections, App admins need to add this escape character when the crontab is actually updated.

0,5,10,15,20,25,30,35,40,45,50,55 \* \* \* \* bash /u01/maximus/maxdat/IL8/scripts/etl\_job\_control.bash 2>&1 >> /u01/maximus/maxdat/IL8/logs/ETLJobControl/etl\_job\_control\_$(date +%Y-%m-%d).log  
00 02 \* \* \* find /u01/maximus/maxdat/IL8/logs/ETLJobControl/ -mtime +30 -exec rm {} \; 2>&1 >> /u01/maximus/maxdat/IL8/logs/ETLJobControl/removed\_$(date +'%Y%m%d\_%H%M%S\_%3N').log  
00 02 \* \* \* find /u01/maximus/maxdat/IL8/data/ -mtime +30 -exec rm {} \; 2>&1 >> /u01/maximus/maxdat/IL8/logs/ETLJobControl/removed\_$(date +'%Y%m%d\_%H%M%S\_%3N').log

# Test

### Through Crontab

Once the process is installed in DEV or UAT using section 3 (installation), process should execute the jobs per JOB\_SCHEDULE defined for each script. The Linux log folders and DB tables (ETL\_JOB\_STATUS and ETL\_JOB\_RUN) can be verified to check the job status.

### Adhoc

ETL Job management process can be triggered manually by executing command from Linux scripts folder. After execution, the Linux log folders and DB tables (ETL\_JOB\_STATUS and ETL\_JOB\_RUN) can be verified to check the job status.

bash etl\_job\_control.bash

# Administration

## Global Configuration

Update ETL\_JOB\_GLOBAL\_CONFIG table as needed. Though the DML to insert global variables is part of etl\_job\_insert\_scripts.sql and can be deployed through CAB, these variables can be changed if needed. The actual retention will happen when PURGE\_ETL\_JOB\_DATA procedure of ETL\_JOB package is executed manually.

* + ETL\_AUDIT\_DATA\_RETENTION\_DAYS should be set to number of days the data in ETL\_JOB\_CONFIG\_AUDIT table should be retained
  + ETL\_RUN\_DATA\_RETENTION\_DAYS should be set to number of days the data in ETL\_JOB\_RUN table should be retained
  + ETL\_LOG\_DATA\_RETENTION\_DAYS should be set to number of days the data in ETL\_JOB\_LOG table should be retained
  + MAX\_PARALLEL\_JOBS\_ALLOWED should be set of maximum number of scripts from ETL\_JOB\_CONFIG that ETL job management process can execute in parallel. If there are more than this many jobs, those jobs will be skipped and picked up in the next run of ETL Job management process
  + JOB\_TIMEOUT\_SEC is the timeout for any job at project level. Any script specific timeout can be set at ETL\_JOB\_CONFIG.JOB\_TIMEOUT\_SEC field. ETL job management will check the current running jobs execution time and compares with these fields (if available), and raise a warning message “job is either stuck or long running…” if the execution time exceeds these fields in seconds.

## Configuring Jobs

Insert/Update ETL\_JOB\_ CONFIG table as needed. Though the DML to inserting job scripts is part of etl\_job\_insert\_scripts.sql and can be deployed through CAB, any necessary changes or new job configurations can be done directly into ETL\_JOB\_CONFIG table. This table captures any delete or update transactions on this table into ETL\_JOB\_CONFIG\_AUDIT table. It is not preferable to delete any job and it should be DISABLED instead based on the package function. ETL job management will skip all disabled scripts from execution.

## ETL\_JOB Package

Only the below 6 procedures are meant for administration purpose. All other procedures and functions of ETL\_JOB packages are either used by the Linux scripts or used internally within the package procedures or functions.

### Procedure ENABLE\_ETL\_JOB

This procedure accepts 2 parameters:

* p\_job\_id : ETL\_JOB\_CONFIG.JOB\_ID should be passed
* p\_apply\_to\_childs : OPTIONAL; Default is N (not applicable for dependent jobs), else Y (applicable for dependent jobs)

|  |
| --- |
| **Actions performed by procedure** |
| ETL\_JOB\_CONFIG.JOB\_ENABLED = Y |

This procedure may be executed with the below syntax.

Here are couple of examples: a) Enable JOB\_ID = 1 b) Enable JOB\_ID = 2 including dependent jobs.

BEGIN

ETL\_JOB.ENABLE\_ETL\_JOB(1);

END;

BEGIN

ETL\_JOB.ENABLE\_ETL\_JOB(2, ‘Y’);

END;

### Procedure DISABLE\_ETL\_JOB

This procedure accepts 2 parameters:

* p\_job\_id : ETL\_JOB\_CONFIG.JOB\_ID should be passed
* p\_apply\_to\_childs : OPTIONAL; Default is N (not applicable for dependent jobs), else Y (applicable for dependent jobs)

|  |
| --- |
| **Actions performed by procedure** |
| ETL\_JOB\_CONFIG.JOB\_ENABLED = N |

This procedure may be executed with the below syntax.

Here are couple of examples: a) Disable JOB\_ID = 1 b) Disable JOB\_ID = 2 including dependent jobs.

BEGIN

ETL\_JOB.DISABLE\_ETL\_JOB(1);

END;

BEGIN

ETL\_JOB.DISABLE\_ETL\_JOB(2, ‘Y’);

END;

### Procedure COMPLETE\_ETL\_JOB

This procedure accepts just 1 parameter:

* p\_job\_id : ETL\_JOB\_CONFIG.JOB\_ID should be passed

|  |  |
| --- | --- |
| **Scenario for the input job** | **Actions performed by procedure** |
| 1) ADHOC 2) INDIVIDUAL 3) PARENT with NO active CHILD jobs | ETL\_JOB\_RUN.RUN\_STATUS = COMPLETED (for the last RUN\_ID) ETL\_JOB\_RUN.RUN\_END\_DT = SYSDATE (for the last RUN\_ID) ETL\_JOB\_STATUS.JOB\_STATUS = COMPLETED ETL\_JOB\_STATUS.JOB\_RUNNABLE = Y |
| 1) PARENT and has active CHILD jobs 2) CHILD and has active CHILD jobs | ETL\_JOB\_RUN.RUN\_STATUS = WAITING (for the last RUN\_ID) ETL\_JOB\_RUN.RUN\_END\_DT = SYSDATE (for the last RUN\_ID) ETL\_JOB\_STATUS.JOB\_STATUS = WAITING ETL\_JOB\_STATUS.JOB\_RUNNABLE = N |
| CHILD with NO active CHILD jobs | ETL\_JOB\_RUN.RUN\_STATUS = COMPLETED (for the last RUN\_ID) ETL\_JOB\_RUN.RUN\_END\_DT = SYSDATE (for the last RUN\_ID) ETL\_JOB\_STATUS.JOB\_STATUS = COMPLETED ETL\_JOB\_STATUS.JOB\_RUNNABLE = Y  Calls UPD\_ETL\_JOB procedure for the job's PARENT\_JOB\_ID with status as COMPLETED |

This procedure may be executed with the below syntax. This example is to COMPLETE a job with JOB\_ID = 1.

BEGIN

ETL\_JOB.COMPLETE\_ETL\_JOB(1);

END;

### Procedure RESET\_ETL\_JOB

This procedure accepts 2 parameters:

* p\_job\_id : ETL\_JOB\_CONFIG.JOB\_ID should be passed
* p\_apply\_to\_childs : OPTIONAL; Default is N (not applicable for dependent jobs), else Y (applicable for dependent jobs)

Note that: RESET is replacing the existing process of removing the check file and resetting the job for immediate execution.

|  |  |
| --- | --- |
| **Scenario for the input job** | **Actions performed by procedure** |
| Only INDIVIDUAL AND PARENT jobs can be RESET. It will set the job to RUNNABLE with JOB\_NEXT\_EXEC\_DT = SYSDATE, so that it will be run in the next ETL Job Control Run | For the job as well as all the child jobs that are running currently, ETL\_JOB\_RUN.RUN\_STATUS = RESET (for the last RUN\_ID) ETL\_JOB\_RUN.RUN\_END\_DT = SYSDATE (for the last RUN\_ID) ETL\_JOB\_STATUS.JOB\_STATUS = RESET ETL\_JOB\_STATUS.JOB\_RUNNABLE = Y  For the job, ETL\_JOB\_STATUS.JOB\_NEXT\_EXEC\_DT = SYSDATE |

This procedure may be executed with the below syntax.

Here are couple of examples: a) Reset JOB\_ID = 1 b) Reset JOB\_ID = 2 including dependent jobs.

BEGIN

ETL\_JOB.RESET\_ETL\_JOB(1);

END;

BEGIN

ETL\_JOB.RESET\_ETL\_JOB(2, ‘Y’);

END;

### Procedure CANCEL\_ETL\_JOB

This procedure accepts 2 parameters:

* p\_job\_id : ETL\_JOB\_CONFIG.JOB\_ID should be passed
* p\_apply\_to\_childs : OPTIONAL; Default is N (not applicable for dependent jobs), else Y (applicable for dependent jobs)

|  |  |
| --- | --- |
| **Scenario for the input job** | **Actions performed by procedure** |
| Only INDIVIDUAL AND PARENT jobs can be CANCELLED. Unlike RESET, process will not update the JOB\_NEXT\_EXEC\_DT and it will let the job executed next based on the JOB\_NEXT\_EXEC\_DT already calculated | For the job as well as all the child jobs that are running currently, ETL\_JOB\_RUN.RUN\_STATUS = CANCELLED (for the last RUN\_ID) ETL\_JOB\_RUN.RUN\_END\_DT = SYSDATE (for the last RUN\_ID) ETL\_JOB\_STATUS.JOB\_STATUS = CANCELLED ETL\_JOB\_STATUS.JOB\_RUNNABLE = Y |

This procedure may be executed with the below syntax.

Here are couple of examples: a) cancel JOB\_ID = 1 b) cancel JOB\_ID = 2 including dependent jobs.

BEGIN

ETL\_JOB.CANCEL\_ETL\_JOB(1);

END;

BEGIN

ETL\_JOB.CANCEL\_ETL\_JOB(2, ‘Y’);

END;

### Procedure PURGE\_ETL\_JOB\_DATA

This procedure accepts no parameters. This will delete data from ETL\_JOB\_CONFIG\_AUDIT, ETL\_JOB\_RUN, and ETL\_JOB\_LOG tables based on global parameters for the project - ETL\_AUDIT\_DATA\_RETENTION\_DAYS, ETL\_RUN\_DATA\_RETENTION\_DAYS, and ETL\_LOG\_DATA\_RETENTION\_DAYS respectively.

This procedure may be executed with the below syntax:

BEGIN

ETL\_JOB.PURGE\_ETL\_JOB\_DATA;

END;

# Glossary

**ETL** – Extract, Transform, and Load

**Process** – A business process described by a requirement spreadsheet. Example: Process Letters

**Project** – A set of business processes used to fulfill contractual obligations for BPM for an entity such as a state benefit program. Example: Illinois Enrollment (ILEB)

**SVN** – Subversion – Software revision control system.

**TBA** –To Be Added – Information to be added later.