|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  | |
|  |  |  | |
|  |  |  | |
|  |  |  | |
| PDC 1.2  User Guide | | | |
|  |  |  | |
|  |  |  | |
|  |  |  | |
|  |  |  | |
|  |  |  | |
|  |  |  | |
|  |  |  | |
|  |  |  | |
|  |  |  | |
|  |  |  | |
|  |  |  | |
|  |  |  | |
|  | |  | |
|  |  |  | |
|  | |  | |
|  |  |  | |

**DOCUMENT PREPARATION INFORMATION**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PROJECT NAME** | **PREPARED BY (PRINT)** | **SIGNATURE** | **DATE PREPARED** | **DOCUMENT ID** |
| PDC |  |  | 28.2.2017 13:38 |  |

**CUSTOMER INFORMATION**

|  |  |  |
| --- | --- | --- |
|  | **CONTACT** | **CUSTOMER ACCOUNT** |
|  |  |  |

**GENERAL INFORMATION**

|  |  |
| --- | --- |
| **PROJECT MANAGER** | **SECURITY CLASSIFICATION** |
|  |  |

**ORIGINATOR INFORMATION**

|  |  |  |  |
| --- | --- | --- | --- |
| **ORIGINATOR** | | **TERADATA ORGANIZATION NAME** | |
|  | |  | |
| **TERADATA ORGANIZATION ADDRESS** | | | |
| NA STRŽI 65, PRAHA 4, 140 62 | | | |
| **PHONE NUMBER** | **FAX NUMBER** | | **E-MAIL** |
|  |  | |  |

**VERSION INFORMATION**

|  |  |
| --- | --- |
| **LAST UPDATED** | **MASTER VERSION LOCATION** |
|  |  |

**CHANGE HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **DATE** | **VERSION** | **CHANGE DESCRIPTION** | **APPROVED BY** |
| 30.10.2011 | 0.01 | Template | Teradata CZ |
| 17.04.2013 | 1.0 | Final | Teradata CZ |
| 03.04.2015 | 1.1 | Final | Teradata CZ |
| 15.02.2017 | 1.1 | Update | Teradata CZ |
| 01.11.2017 | 1.2 | Final | Teradata CZ |

###### Table of Contents

[1. Summary 7](#_Toc498352942)

[1.1 Abbreviations, Terms and Definitions 7](#_Toc498352943)

[1.2 Related Documents 7](#_Toc498352944)

[2. What is PDC 8](#_Toc498352945)

[2.1 Concept definition 8](#_Toc498352946)

[2.1.1 Job 9](#_Toc498352947)

[2.1.2 Stream 10](#_Toc498352948)

[2.1.3 Step 10](#_Toc498352949)

[2.1.4 Dependency 10](#_Toc498352950)

[2.1.5 Calendar 10](#_Toc498352951)

[2.1.6 Table lock 11](#_Toc498352952)

[2.1.7 Job priority 11](#_Toc498352953)

[2.1.8 Parallelism 11](#_Toc498352954)

[2.1.9 Job status 11](#_Toc498352955)

[3. PDC Architecture 13](#_Toc498352956)

[3.1 Engine 13](#_Toc498352957)

[3.2 Scheduler 14](#_Toc498352958)

[3.3 Framework checker 14](#_Toc498352959)

[3.4 GUI Application 14](#_Toc498352960)

[4. Engine 15](#_Toc498352961)

[4.1 Engine Scripts 15](#_Toc498352962)

[4.1.1 Engine.pl 15](#_Toc498352963)

[4.1.2 Prepare\_job 16](#_Toc498352964)

[4.1.3 Run\_job 16](#_Toc498352965)

[4.2 Metadata repository procedures 16](#_Toc498352966)

[4.2.1 Procedure SP\_ENG\_GET\_JOB\_LIST 16](#_Toc498352967)

[4.2.2 Procedure SP\_ENG\_UPDATE\_STATUS 17](#_Toc498352968)

[4.2.3 Procedure SP\_ENG\_CHECK\_WD\_STATUS 18](#_Toc498352969)

[4.2.4 Procedure SP\_ENG\_UPDATE\_WD\_STATUS 18](#_Toc498352970)

[4.3 Systems 18](#_Toc498352971)

[4.4 Parallelism control 19](#_Toc498352972)

[4.4.1 Toughness 19](#_Toc498352973)

[5. Scheduler 21](#_Toc498352974)

[5.1 Initialization 21](#_Toc498352975)

[5.2 Small Initialization 22](#_Toc498352976)

[5.3 Big Initialization 23](#_Toc498352977)

[6. Framework checker 25](#_Toc498352978)

[6.1 Checker Script 25](#_Toc498352979)

[6.2 Metadata repository procedures 25](#_Toc498352980)

[6.2.1 Procedure SP\_FWRK\_CHECK\_WD\_STATUS 25](#_Toc498352981)

[6.2.2 Procedure SP\_FWRK\_CHECK\_SCHED\_STATUS 25](#_Toc498352982)

[6.2.3 Procedure SP\_FWRK\_CHECK\_NOTIFICATION 25](#_Toc498352983)

[6.2.4 Procedure SP\_FWRK\_CHECK\_INITIALIZATION 26](#_Toc498352984)

[6.2.5 Procedure SP\_FWRK\_MESSAGE\_GEN 26](#_Toc498352985)

[7. GUI application 27](#_Toc498352986)

[7.1 Logon Page 27](#_Toc498352987)

[7.2 Page Layout 28](#_Toc498352988)

[7.2.1 Status Bar 29](#_Toc498352989)

[7.2.2 Main Menu 29](#_Toc498352990)

[7.2.3 Monitor Header 29](#_Toc498352991)

[7.2.4 Status line 30](#_Toc498352992)

[7.3 DASHBOARD 30](#_Toc498352993)

[7.4 Monitor 32](#_Toc498352994)

[7.4.1 Monitor – Stream View 34](#_Toc498352995)

[7.4.2 Monitor – Job View 35](#_Toc498352996)

[7.4.3 Monitor – Running Jobs 35](#_Toc498352997)

[7.4.4 Monitor – Failed Jobs 36](#_Toc498352998)

[7.4.5 Monitor – Ready to Run Jobs 37](#_Toc498352999)

[7.4.6 Monitor – Finished Jobs 38](#_Toc498353000)

[7.4.7 Monitor – All Jobs 40](#_Toc498353001)

[7.4.8 Monitor – Odd Jobs 40](#_Toc498353002)

[7.5 Control 42](#_Toc498353003)

[7.5.1 Control – Scheduler 42](#_Toc498353004)

[7.5.2 Control – Finish Jobs 43](#_Toc498353005)

[7.5.3 Control – Restart Jobs 44](#_Toc498353006)

[7.5.4 Control – Restart Finished Jobs 45](#_Toc498353007)

[7.5.5 Control – Block Jobs 46](#_Toc498353008)

[7.5.6 Control – Unblock Jobs 47](#_Toc498353009)

[7.5.7 Control – Manual Batch 48](#_Toc498353010)

[7.6 Logs 51](#_Toc498353011)

[7.6.1 Logs - GUI\_LOG\_CTRL\_ACTION 51](#_Toc498353012)

[7.6.2 Logs - STAT\_LOG\_MESSAGE\_HIST 52](#_Toc498353013)

[7.6.3 Logs - STAT\_LOG\_EVENT\_HIST 53](#_Toc498353014)

[7.7 SESS TABLES 53](#_Toc498353015)

[7.8 CTRL TABLES 53](#_Toc498353016)

[7.8.1 CTRL TABLES – Stream 54](#_Toc498353017)

[7.8.2 CTRL TABLES – Job 54](#_Toc498353018)

[7.8.3 CTRL TABLES – Stream Dependency 55](#_Toc498353019)

[7.8.4 CTRL TABLES – Job Dependency 56](#_Toc498353020)

[7.8.5 CTRL TABLES – Stream Plan Ref 56](#_Toc498353021)

[7.8.6 CTRL TABLES – Job Table Ref 57](#_Toc498353022)

[7.9 Admin 57](#_Toc498353023)

[7.9.1 Admin – Label maintenance 57](#_Toc498353024)

[7.9.2 Admin – Change Management 58](#_Toc498353025)

[7.9.3 Admin – Access Control 59](#_Toc498353026)

[7.10 Reports 60](#_Toc498353027)

[8. Metadata repository tables 61](#_Toc498353028)

[8.1 CTRL tables 61](#_Toc498353029)

[8.1.1 CTRL\_STREAM table 61](#_Toc498353030)

[8.1.2 CTRL\_STREAM\_DEPENDENCY table 61](#_Toc498353031)

[8.1.3 CTRL\_STREAM\_PLAN\_REF table 61](#_Toc498353032)

[8.1.4 CTRL\_JOB table 61](#_Toc498353033)

[8.1.5 CTRL\_JOB\_DEPENDENCY table 62](#_Toc498353034)

[8.1.6 CTRL\_JOB\_TABLE\_REF table 62](#_Toc498353035)

[8.2 SESS tables 63](#_Toc498353036)

[8.2.1 SESS\_JOB (SESS\_JOB\_BCKP) table 63](#_Toc498353037)

[8.2.2 SESS\_JOB\_DEPENDENCY (SESS\_JOB\_DEPENDENCY\_BCKP) table 63](#_Toc498353038)

[8.2.3 SESS\_QUEUE table 64](#_Toc498353039)

[8.2.4 SESS\_STATUS table 64](#_Toc498353040)

[8.2.5 SESS\_JOB\_TABLE\_REF table 64](#_Toc498353041)

[8.2.6 SESS\_PARALLELISM\_CONTROL 65](#_Toc498353042)

[8.3 PDC config tables 65](#_Toc498353043)

[8.3.1 CTRL\_JOB\_STATUS table 65](#_Toc498353044)

[8.3.2 CTRL\_NBD\_CAL table 65](#_Toc498353045)

[8.3.3 CTRL\_NEXT\_STATUS table 65](#_Toc498353046)

[8.3.4 CTRL\_TASK\_PAREMETERS table 66](#_Toc498353047)

[8.3.5 CTRL\_PARAMETERS table 66](#_Toc498353048)

[8.4 Framework checker tables 66](#_Toc498353049)

[8.4.1 CTRL\_NOTIFICATION table 66](#_Toc498353050)

[8.4.2 CTRL\_NOTIFICATION\_TYPES table 67](#_Toc498353051)

[8.4.3 CTRL\_NOTIFICATION\_RLTD table 67](#_Toc498353052)

[8.4.4 CTRL\_NOTIFICATION\_SEVERITY table 67](#_Toc498353053)

[8.5 Statistics and events 67](#_Toc498353054)

[8.5.1 STAT\_STATUS table 67](#_Toc498353055)

[8.5.2 STAT\_JOB\_STATISTICS table 67](#_Toc498353056)

[8.5.3 SESS\_JOB\_STATISTICS table 68](#_Toc498353057)

[8.5.4 STAT\_LOG\_EVENT\_HIST table 68](#_Toc498353058)

[8.5.5 STAT\_LOG\_MESSAGE\_HIST table 69](#_Toc498353059)

[8.6 Lookups 69](#_Toc498353060)

[8.6.1 LKP\_APPLICATION table 69](#_Toc498353061)

[8.6.2 LKP\_COUNTRY table 69](#_Toc498353062)

[8.6.3 LKP\_ERROR\_CD table 69](#_Toc498353063)

[8.6.4 LKP\_JOB\_CATEGORY table 70](#_Toc498353064)

[8.6.5 LKP\_JOB\_TYPE table 70](#_Toc498353065)

[8.6.6 LKP\_PHASE table 70](#_Toc498353066)

[8.6.7 LKP\_PLAN table 70](#_Toc498353067)

[8.7 GUI tables 70](#_Toc498353068)

[8.7.1 GUI\_CHANGE\_MANAGEMENT table 70](#_Toc498353069)

[8.7.2 GUI\_CHANGE\_CONTROL table 70](#_Toc498353070)

[8.7.3 GUI\_LOG\_CTRL\_ACTION table 71](#_Toc498353071)

# Summary

The document is a user guide for PDC product. It describes each component of PDC application, the component design, functionality and relation.

## Abbreviations, Terms and Definitions

| Abbreviation | Description |
| --- | --- |
| PDC | Process Dispatch Center |
| PID | Process identificator – identification of running process within operating system |
| JOB | Controled and monitored task (Teradata, SSH, ETL, CMD …) |
| STREAM | Couple of related jobs |
| STEP | Job’s part |
| ENGINE | Executive part of PDC application which lanches jobs |
| SCHEDULER | Hand-over of jobs to the Engine |
| METADATA | Storage of configuration and opareting data and procedure logic |
| GUI | Graphical User Interface for controlling and monitoring |
| CHECKER | PDC self checking application |
| SNMP | Simple Network Management Protocol |
| MANUAL BATCH | Solicit processing of some jobs |

## Related Documents

| ID | Name | Description |
| --- | --- | --- |
| 1 | PDC\_Installation\_Guide\_v1.docx | Installation step by step guide |
| 2 |  |  |

# What is PDC

Process Dispatch Centre (PDC) is controlling framework application, which is used for optimization of ETL/ELT process and its supporting tasks. The main task of PDC is launching separate jobs in sequence based on their dependencies and priorities. While doing so it utilizes optimally systems resources and uses parallelism as much as possible. The frameworks logic takes care about optimal launch of all jobs, while the internal logic of the job itself is invisible to the framework. Framework runs all jobs in the same manner – using a child process, which interprets the content of the job command line – and waits for the job exit code. In dependency on the exit code the framework considers the job finished either successfully or failed.

All necessary information needed by the framework is stored as metadata in Oracle database.

Metadata are used for:

• job configuration

• operational information about current and historical behavior of the jobs

• interface between all PDC parts

• source for representation of the current job status at GUI interface

• controlling of job behavior

There is no limitation of job type which can be executed by PDC application; the only limitation is that the interface used for job launching has to be installed on system where the framework Engine is running (e.g. for running an Informatica Job the Informatica Client need to be present on the system).

## Concept definition

Tasks controlled by PDC application belongs to three hierarchical levels:

|  |  |
| --- | --- |
| STREAM | Virtual class joining jobs which together take care about some specific part of processing. The launching of jobs on the stream level is controlled using the calendar functionality. |
| JOB | The most important level is the job level. PDC application performs all controlling work on this level the, and job represents separate lowest level of a launch-able task. All configuration information is directly or indirectly linked to job level. |
| STEP | Breakdown of jobs. Steps are invisible from PDC application point of view. Step is subpart of a job and all steps of a job are running in a sequence. Step task is to provide specific functionality as load data into database, perform transformation and so on. To allow correct parallelism, control the job can contain only steps of similar task type. Even if steps are for PDC application invisible, PDC can work with steps in case of job restart after failure – as resume functionality enables continuation of job processing from a failed step. |

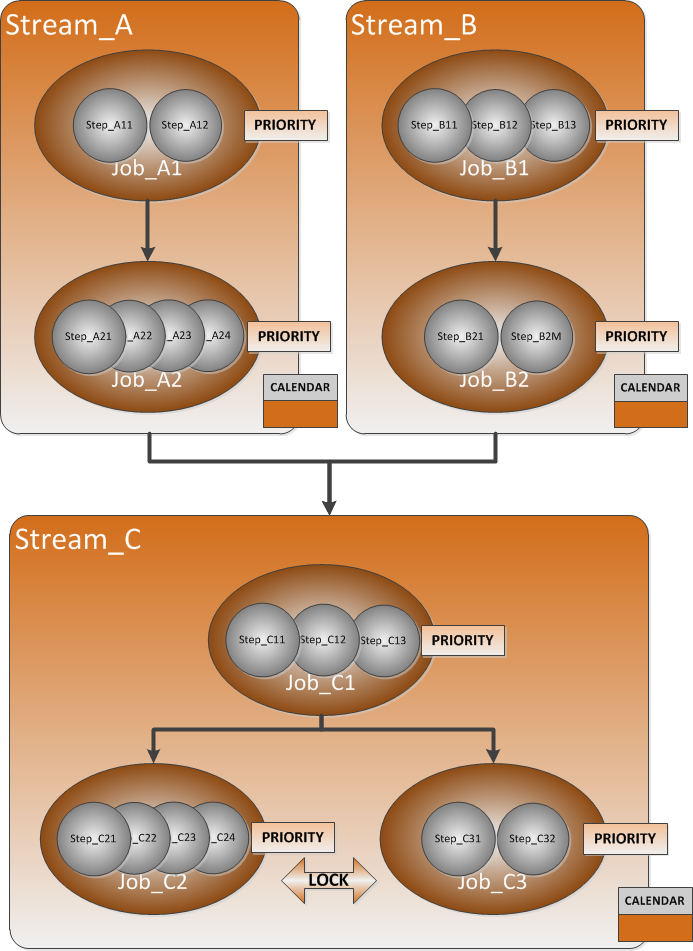


Figure 1 – streams, jobs and steps relations

Picture above shows the main relations between streams, jobs and job’s steps. Dependency on stream level as well as dependency on job level is shown. Priority of the job is defined on job level contrary the calendar, which is defined on stream level respecting the fact that all jobs within stream should be launched or be skipped synchronously. Table lock on job level increases efficiency of job processing by blocking competition jobs common run.

### Job

Job is basic build part, which represents separate launch-able task performing demand activity. The job sequence represents processing. From PDC perspective the job is the main build stone of processing. All information stored in PDC metadata is directly or indirectly pointed to a job. From the point of launch ability, the job can occur in two states: run able job and skip able job. This initial state is assign to the job in time of initialization. This state is coming from the state of the stream in which is job located. The job name is unique identification of the job and must be unique within all Engines. Every job is related to only one instance of the Engine and can be located only in one stream.



Figure 2 - stream

Picture above discloses the fact that Scheduler physically cannot work on stream level. Technically all streams are degraded into job level using auxiliary jobs STREAM\_BEGIN and STREAM\_END and adding necessary dependency between physical and auxiliary jobs (red lines).

### Stream

Stream is virtual class joining jobs, which together take care about some specific part of processing. The launching order of stream in processing is defined by stream to stream dependencies such as launching order of the jobs inside the stream is defined by job to job dependencies. All jobs inside of the stream have the same launching schema therefore it is controlled using the calendar functionality on stream level. There is no relation on Engine instance on stream level. This relation is taken from jobs collected in stream. Therefore the stream can contain only jobs with the same value of engine\_id.

### Step

Job can consist of one or more steps. From PDC point of view, the steps are invisible therefore PDC is making all control on job level only. But it doesn’t mean that steps are unusable. PDC uses steps for providing restart and resume functionality. Which steps can be put together within one job is defined by a rule that all steps inside one job has to have the same execution type (e.g. Oracle, Informatica transformation, Teradata load).

### Dependency

PDC application, Scheduler specially, chooses launch-able jobs for processing by Engine. During job selection Scheduler has to respect many rules, the basic rule is job to job dependency which represents processing data flow. In metadata there the dependencies are defined in special table as child to parent relation. We recognize relation between jobs inside the stream and the relation between streams.

### Calendar

Not all jobs are requested to launch every day, some jobs have to launch only on weekly or monthly bases. Calendar gives us possibility to control it. It is not very usual that one job can solve whole task, more often a group of job does it. From this reason the calendar is not defined on job level, but calendar is defined on stream level. Practically it means that in specific day all jobs in stream are all launched or all jobs are skipped. There is no limitation on how many calendar can be assign to single stream. The requests of each calendar are compared and if all calendar wont’s run job, the job is skipped otherwise all jobs are launched.

### Table lock

Against calendar the table lock is defined on job level. The lock says which tables in which databases are operated by a job. Main reason for having this functionality is protecting to launch competition job is same time. Typically, we recognized two types of the lock – write lock and read lock. There are the same operational rules in using locks as in database is. If two jobs have read lock on the same table, these jobs can run simultaneously, but if some job has write lock, it has to run alone its competitors.

### Job priority

Every job has own priority. The priority is used for job selecting for processing when several jobs is prepared for run but only some can be run now. In this case the job with higher (lower number) priority is choose. The default value of priority is 1000, the lower number is 0, but zero priority is reserved for system tasks only (as abort job) and can‘t be used for regular job therefore jobs with zero priority don’t respect other rules such as rule for parallel control and so on.

### Parallelism

Parallelism is controlled by limitations imposed on total number of jobs running in parallel together with limitations on concurrency of jobs belonging to the same categorization (category and subcategory).

### Job status

Every job must belong to specific state called status. The value of status says in which state job currently is and also in which statues job was in history. Statuses can be assigning to classes. The main reason has three classes:

* Prepared for run class which include jobs prepared for launch or skip
* Running class, which include jobs which are currently processing
* Finished class includes jobs which already finished its processing.

The status change is done by signals. There are defined many signals, each signal has special meaning such as launching of job, blocking of job, finishing job successfully or failed job and so on.

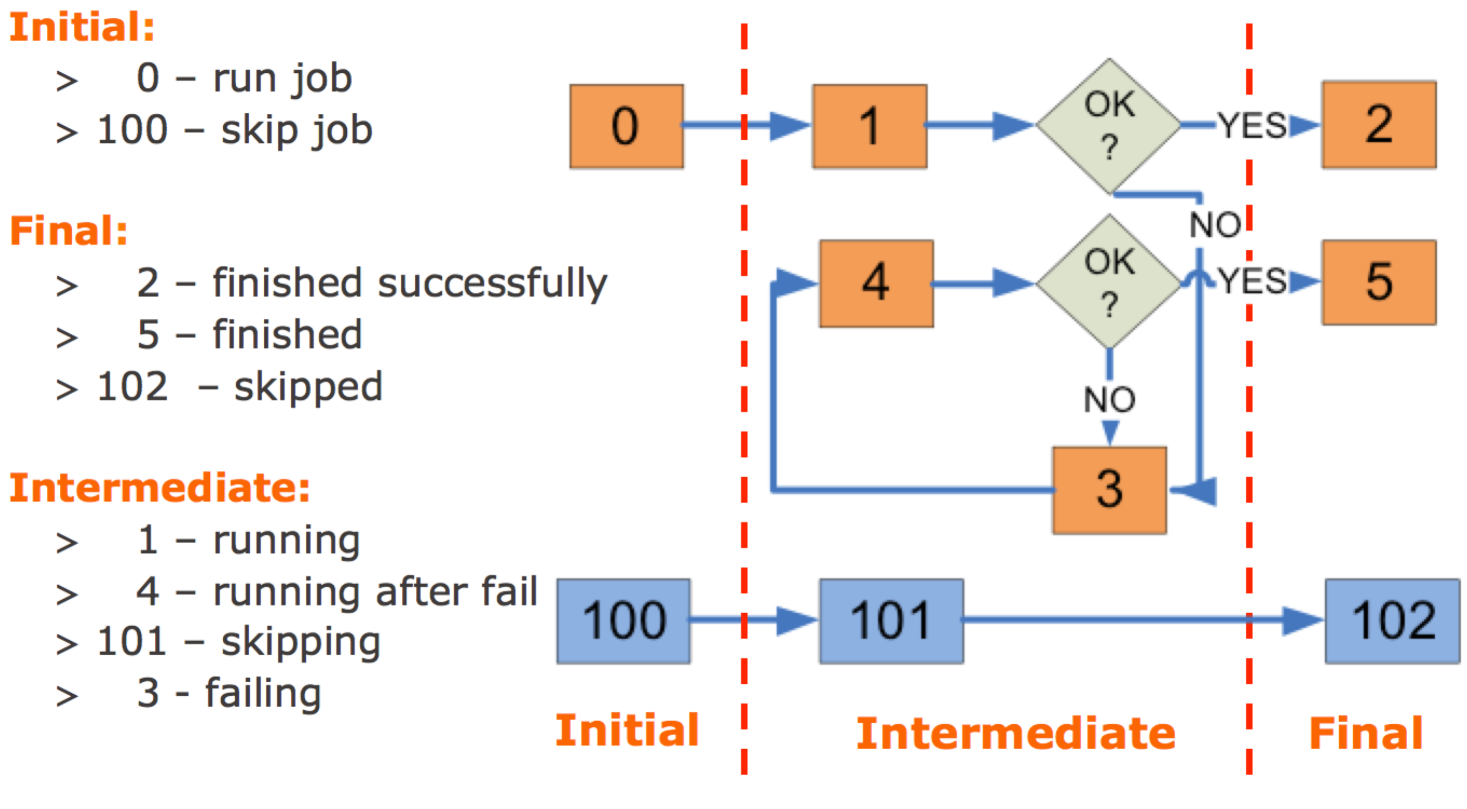


Figure 3 - job state flowchart diagram

Picture shows basic status movement during job’s life cycle. Depending on calendar initially job starts in status 0 – prepared for run or in status 100 – prepared for skip (job will not be launched today). Every job must be finished first, otherwise its child cannot be launched. The most typical status changes are from status 0 over 1 to 2 for running job or from 100 over 101 to 102 for skipping job. When job is launched, jobs status is changed from 0 to 1. If something wrongs happened during job’s run, job reaches status 3 instead of status 2. After some delay, job is launched again, which changed the job’s status to 4. There is no difference between status 1 and status 4, status 4 only “remember” job’s problem on its path. When job is finished successfully it reach status 2 (finished successfully) or status 5 (finished after some errors), which is signal for supervisor to check what wrong happened during job’s run.

# PDC Architecture

The basic concept of PDC application family is shown in the picture below:

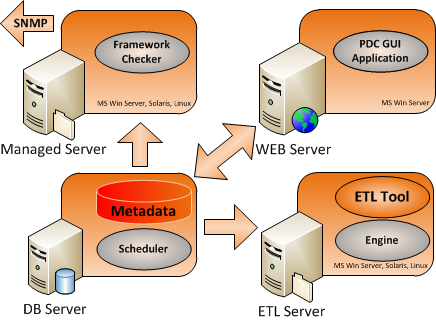


Figure 4 - PDC component architecture

PDC application uses for its work several parts – each taking care about specific functionality:

|  |  |
| --- | --- |
| ENGINE | Executive part which launches jobs and tracks their run |
| SCHEDULER | Hand-over of jobs to the Engine |
| METADATA REPOSITORY | Interdependency logic stored in Oracle procedures as well as configuration data |
| GUI | Front end, GUI enables all necessary type of actions with PDC application.   * Monitoring - tracking progress of processing and controlling job work. * Controlling metadata creation * Change management support and related tasks. |
| CHECKER | Checker detects a suspicious or erroneous situation it sends a SMTP trap to the alerting system. |

## Engine

Engine represents an executive part of control framework. The main Engine responsibility is job execution and operational metadata update. The Engine launches job as independent child process which runs for whole job’s life cycle and when job has finished, update job status in metadata table. Engine is something like never ending cycle which gets launchable jobs from Scheduler, launching them and waiting for next job for execution. Engine starts form system scheduler typically every 5 minutes, but only one Engine can run for specific engine\_id. When new instance of Engine starts, it checks the old Engine functionality. If old Engine seems to be running, the new instance stops its run, otherwise kill the process of the old Engine and takes control.

## Scheduler

Scheduler represents the brain of control framework. The procedures and tables, which are a part of Scheduler, contain almost all-processing logic. The main task of Scheduler is handing information about jobs to the Engine. Due this functionality the Scheduler is taking a care about all job-related task, about parallelism control, about information delivery into all supporting application and also prepares data for Framework checker. Scheduler enables temporary stops job’s processing in case of maintenance ask. There is one instance of Scheduler for every instance of Engine.

## Framework checker

Framework checker is independent application that is responsible for checking of PDC work and alerting every irregularity. Checker is scheduled in CRON typcially runs every five minutes. Application owns its private metadata, which contains information about typical behavior of every checked job. Application in every cycle compares current state of job with typical state and all irregularity, fail and so on reports using SMNP trap on console. Checker is looking for of these events:

• Engine stops

• Scheduler stops when some jobs are prepared for launch

• Job reaches required status

• Job doesn’t reach required status in defined time

• Job is running longer then is expected

Checker doesn’t report jobs which are running longer than expected immediately when the state appears, but allow defining some hysteresis which is protecting reporting an error on job running 6 seconds with typical run duration 5 seconds. Framework checker is not dedicated for each instance of Engine, but one Checker checks behavior of all Engines, Schedulers and jobs.

## GUI Application

Metadata permits PDC application administers all parts of its functionality. GUI application is WEB based application supporting standard browser functionality and using Teradata Viewpoint graphical layout. The user must have appropriate permissions for working with any part of the application. The main ask on GUI application is to enable comfortable monitoring and controlling of job processing, but GUI application is not necessary for PDC work. Jobs can be located in different instances of PDC; these instances are totally independent and are used for controlling and monitoring independent data processing.

# Engine

Engine represents an executive part of control framework. Engine is consisting of three PERL scripts:

* Engine.pl – in cycle asks Scheduler for jobs prepared for processing. For every job launch Prepare.pl script
* Prepare\_job.pl – makes Run\_job.pl script independent on Engine.pl. Simply Prepare\_job.pl launches Run\_job.pl and dies.
* Run\_job.pl – launches real job and waits

There are several steps in job processing:

* When Engine starts primary it has to test previous Engine status. In case when previous Engine works correctly, the new one dies, otherwise it kills previous Engine and takes over its work.
* Engine calls Scheduler to receive a list of runnable jobs for processing
* For every job from the list Engine calls Oracle procedure SP\_ENG\_UPDATE\_STATUS, which changes job status from prepared to running.
* Engine starts Prepare\_job.pl which transfers control on job to Run\_job.pl
* Run\_job.pl physically launches job and waits for job exit code. When job has finished Run\_job.pl calls procedure SP\_ENG\_UPDATE\_STATUS which changes job status using job exit code. In case of success, procedure SP\_ENG\_UPDATE\_STATUS deletes job’s child dependency.
* When all jobs from list are processed, Engine calls procedure SP\_ENG\_UPDATE\_WD\_STATUS which saves the timestamp of cycle for Engine work recognition.

## Engine Scripts

### Engine.pl

Engine.pl is started from CRON (recommend interval is five minutes). There are several instances of Engine.pl available at same time. Every Engine instance is taking care of jobs with defined engine\_id value.

Engine uses these Oracle procedures:

* SP\_ENG\_GET\_JOB\_LIST – job list creation for processing
* SP\_ENG\_UPDATE\_STATUS – writes status change into SESS\_JOB and SESS\_STATUS table
* SP\_ENG\_CHECK\_WD\_STATUS – decides if old Engine working correctly
* SP\_ENG\_UPDATE\_WD\_STATUS – writes CURRENT\_TIMESTAMP of last Engine cycle into CTRL\_PARAMETERS table

Engine main steps:

1. Engine initializes log file \_\_\_Engine\_@<system\_name>@<engine\_id>@<start\_timestamp>.log
2. Program calls procedure SP\_ENG\_CHECK\_WD\_STATUS to decide if previous Engine is running
3. If previous Engine is recognized as dead, program takes PID of previous Engine from PID<engine\_id>\_<system\_name>.PID file and kills it. After that writes own PID into this file and continue in work.
4. Engine calls procedure SP\_ENG\_UPDATE\_WD\_STATUS to write current timestamp into CTRL\_PARAMETERS table.
5. Procedure SP\_ENG\_GET\_JOB\_LIST represents Scheduler functionality. There are these possible outputs from procedure:
   * + List of jobs and asked activity (run, restart, resume)
     + No job and status = 8 which means that no job is possible to run now (wait cycle)
     + No job and status = 9, which means that all jobs are already processed, Engine stops.
6. In case when run, restart or resume of job is required, procedure SP\_ENG\_UPDATE\_STATUS is called only once for status changing to running
7. Next the program Prepare\_job.pl gives control to Run\_job.sh which is taking care for job during whole job’s run. If launching of Prepare\_job failed, Engine calls procedure SP\_ENG\_UPDATE\_STATUS once more to change job’s status from running to failing.
8. When Prepare\_job.sh is successfully launched the Engine doesn’t care about job more.
9. When all jobs from list are processed, Engine returns to the fifth point.

### Prepare\_job

Program Prepare\_job.pl makes Run\_job.pl independent on Engine.pl.

### Run\_job

Program Run\_job launches job and waits for job’s exit code. When job is finished it calls procedure SP\_ENG\_UPDATE\_STATUS and gives it job’s exit code. Depending of exit code the job is finished when exit code equal zero or failed when not zero exit codes is received. Some job type can have extended exit code evaluation. In this case job can reach also other statuses, not only finished or failed.

## Metadata repository procedures

### Procedure SP\_ENG\_GET\_JOB\_LIST

Procedure is called periodically from Engine to allow Engine obtains list of runnable jobs. Procedure also returns signal which can stop or wait the Engine. Generally said the procedure represents complex Scheduler functionality. All asks for parallel control, dependency and lock respect are controlled by this procedure.

Engine gives value of engine\_id parameter when calls procedure.

Procedure returns:

* Availability = 9 – all jobs for engine\_id are already finished, Engine stops
* Availability = 8 – there are no jobs for processing at this time, Engine waits
* Availability = 1 – job will started
* Availability = 2 – job will be restarted
* Availability = 3 – job will be resumed

Temporally the list of jobs prepared for launch is placed in TEMP\_ENG\_JOB\_READY table.

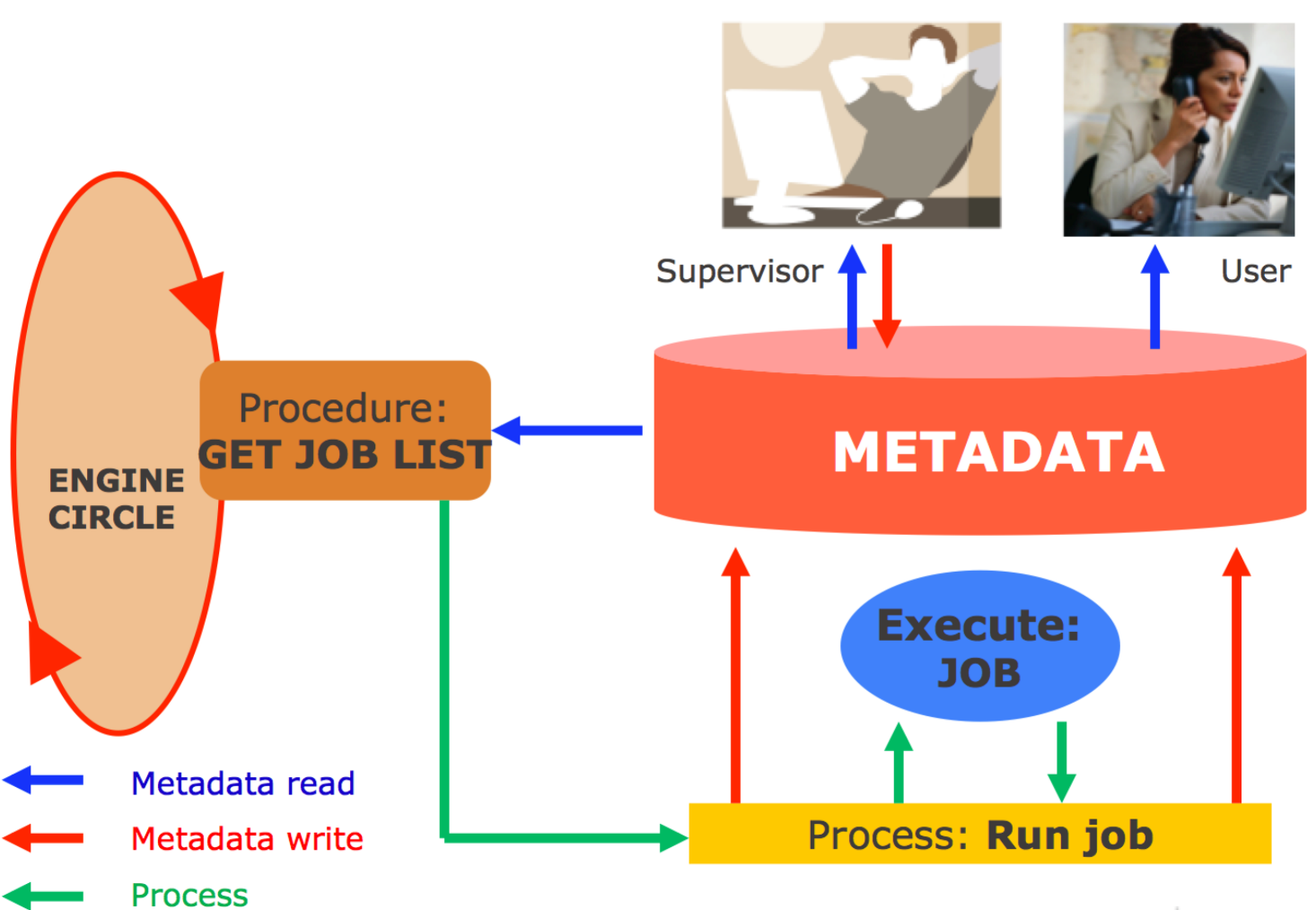


Figure 5 - Engine conceptual architecture

Picture shows the basic Engine concept. Engine in fact is only a program circle, which in every round calls procedure to get list of jobs to be launched. For every job dedicated thread is created. Thread process launches job’s command line and wait for job’s exit status. Metadata are populated by Engine before job starts (status 0 to status 1, …) and also by thread process when job finish.

### Procedure SP\_ENG\_UPDATE\_STATUS

Procedure changes job status in SESS\_JOB table and writes them also into SESS\_STATUS log table. Procedure run differ on current job state - if jobs has been started or finished. These parameters are used for procedure call:

* Job\_id – job unique identifier
* Request – asked change type
* Launch – if job is started or finished
* Queue\_number – queue number used for job run
* Engine\_id – Engine identifier

Request is transferred into signal inside the procedure using value “launch” and cont\_anyway (success of job finish is asked or not). Currently these requests are used:

* BLOCK – ask for job blocking or unblocking
* SUCCESS – job has been successfully started or finished
* FAILED – unsuccessfully finished job
* MARK\_FINISHED – forcedly finished job

The rule details are stored in CTRL\_NEXT\_STATUS table. The status description and also status grouping can be found in CTRL\_JOB\_STATUS table.

After start the new status of the job is taken from CTRL\_NEXT\_STATUS table using input parameters. The value of job status and last\_update is updated in SESS\_JOB table and the new job state is also inserted into SESS\_STATUS log table. Also queue is occupied or released in SESS\_QUEUE table depend of job launch value.

### Procedure SP\_ENG\_CHECK\_WD\_STATUS

Procedure is used for detection of Engine runs. The time difference between current time and time of the last cycle, which is taken from the WATCHDOG\_STATUS in CTRL\_PARAMETERS table, is calculated. The result is compared with WATCHDOG\_INTERVAL in CTRL\_PARAMETERS table. If the result is less or equal then mentioned value, the current Engine is evaluated as running otherwise is evaluated as dead and new Engine will replace it.

### Procedure SP\_ENG\_UPDATE\_WD\_STATUS

Procedure is used for record of Engine cycle. Current timestamp is stored into WATCHDOG\_STATUS in CTRL\_PARAMETERS table.

## System

Before we can discuss “system feature” advantages, we have to be sure that the difference between Engine and Scheduler is clear.

Scheduler is independent process, which has no direct dependency on other Schedulers. If we have several independent processing like Daily batch, Inter-day stage filling or Weekly metadata cleaning process, it is good way to use separate Scheduler for this tasks. Every Scheduler has unique value of engine\_id, which defines which Engine launches jobs of this Scheduler.

Engine is process, which is processing jobs dedicated to “its” Scheduler. If we have more Engines for one Scheduler, the Engines depends each other. It means that if one Engine is asking Scheduler for a job list for launching, another Engines of this Scheduler have to wait until the first Engine finishes launching of its entire jobs.

System feature means that several (not limited on two only) Engines are processing jobs of one Scheduler. This is useful when building cluster solution when several identical ETL servers are launching the same job coalition. But system feature is also useful, when we have several ETL servers, but every server is dedicated only for special work type. In this case system concept solution allows run this task type locally on suitable server without necessity of calling remote command.

Adding additional Engine to existing environment represents necessity of its announcement in CTRL\_PARAMETERS table and settings its task limits in CTRL\_TASK\_PARAMS table for every server hosting Engine process, which should launch this task type.

## Parallelism control

### Toughness

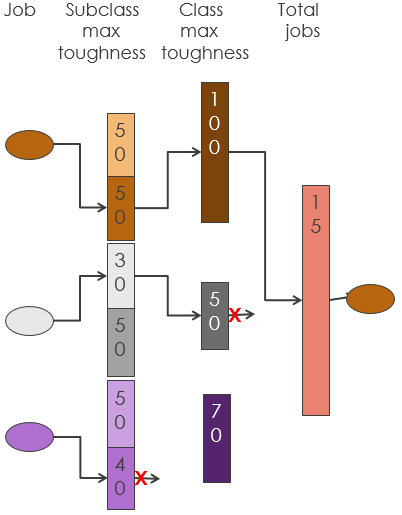


Figure 6 - parallelism control

Parallelism control is one of the most complex processes in PDC logic. Before we can dive deep into whole process, try to explain the principle on the simple example. This situation is described on the picture above. Imagine three walls which block the job goes through. There are holes in the walls, same holes are already occupied by running jobs and some holes are still empty. The numbers written on wall represent number of holes in it. When a new job is chosen as launch-able candidate, its clearness through walls has to be validating first. Each job is determined to go through dedicated wall by its job\_category value. If job successfully passed the first wall, the second and finally the third wall are also tested. When clearness fails on some walls, next job is tested, until all possible launch-able jobs are chosen.

The “walls” limits are defined in CTRL\_TASK\_PARAMS table and limit consumption is represent by SESS\_PARALELLISM table. Each limit is defined separately for engine\_id and server host name or for all engines or systems (all = null value in config field). Value of CTRL\_JOB.job\_category is mapped to CTRL\_TASK\_PARAMETERS.task\_type/SESS\_PARALLELISM\_CONTROL.job\_category (the first wall) and CTRL\_TASK\_PARAMETERS.task\_type/SESS\_PARALLELISM\_CONTROL.job\_category to CTRL\_TASK\_PARAMETERS.parent\_task\_type/SESS\_PARALLELISM\_CONTROL.parent\_job\_category (the second wall). The third wall, which is common for all job’s categories, allows maximal number of concurrently running jobs and is valid for whole Scheduler. If it is lowered to “0” value, factually it stops the Scheduler, which means that any new jobs will be started.

However all jobs aren’t the same, each job has different request of resource consumption. Do not mention that resource amount of each job can be specified separately, generally we recognize four category of resource consumption for typical jobs: SMALL, MEDIUM, LARGE and EXCEPTIONAL, which represents amount of resource consumption valued of 10, 20, 40 and 80 (recommended values). We called this amount as “toughness”. Job’s toughness of every job is defined in CTRL\_JOB table eponymous column.

The “wall” limit in environment, where we are working with job’s toughness, means the summary of all hole’s sizes. If we are starting with the new job\_category, we suggest to set up the toughness initial value equal 150. It allows running one exceptional job or three large jobs or 7 medium or 15 small jobs or its combination.

Event if it can sound well, this principle will not work in practice efficiently. Think first why we suggest using initial value equal 150? When toughness of exceptional job is 80, value 150 doesn’t allow running two exceptional jobs together, but allows with one exceptional job running another large job plus one medium and one small job. Is it enough? We have to say – no. The toughness equal 150 allows concurrently run of 15 small jobs, but in this example only 4 jobs (1 exceptional + 1 large + 1 medium + 1 small) are running. This setting lowers parallel efficiency of this category to one quarter only. Typically exceptional and large jobs are running for very long time period. When we appreciate that every finished jobs unlocks another jobs for processing, the agglomeration of exceptional and large jobs running together decrease significantly processing throughput. To prevent this situation apparence, there is another parameter, which define the minimal amount of jobs that can fulfill category toughness limit. We usually set up the limit on value 6. Practically it means when exceptional job is running we have to fulfill toughness 70 by at least 5 jobs. This preventing launching large job (we need 40 for launch large job, so we can start 4 jobs in remaining 30) and only medium or small jobs will be executed.

Will it work now? We have to say no once more. Imagine the situation when large amount small jobs are running and high priority exceptional job is waiting for launching. In every Engine cycle some small job is finished so it free category toughness valued of 10, 20, or little more, which is suitable for small or medium jobs, but we need 80 for exceptional job launching. If all other criteria for exceptional job launch are met and only insufficient toughness is the cause why job cannot be launched, something similar to “reservation system” is initiated. This blocks small jobs launching in free toughness.

Knowledge of these principles is necessary for correct toughness thresholds settings. These limits cannot be set during several processing, typically it takes one or two months until the limits are effectively defined. During this tuning we track resource consumption and running time of “hard” jobs to categorize them into adequate job category. The next step is monitoring each system running characteristics and changing limits to reach state when system is optimally loaded but not overloaded by running jobs. This is not easy task, but well tuned limits can significantly improve processing efficiency and shortage processing time.

# Scheduler

## Initialization

Initialization is used for setting jobs into initial state. Control metadata are taken from tables:

* CTRL\_STREAM – stream definition – stream\_begin and stream\_end jobs definition
* CTRL\_JOB – job parameters
* CTRL\_STREAM\_DEPENDENCY – stream to stream dependency and dependency type
* CTRL\_JOB\_DEPENDENCY – job to job dependency and dependency type
* CTRL\_STREAM\_PLAN\_REF – calendar for job’s run definition on stream level
* CTRL\_JOB\_TABLE\_REF – table lock definition on job level

Data are stored into tables:

* SESS\_JOB – current state of jobs
* SESS\_JOB\_DEPENDENCY – job to job dependency (parent – child)

There are two steps of initialization. In the first step (small initialization) only that jobs which provided the initialization work are initialized. These jobs can be simply recognized therefore they have phase = INITIALIZATION. In the second step (big initialization) Engine runs the small initialization jobs that are doing all necessary steps for initialization.

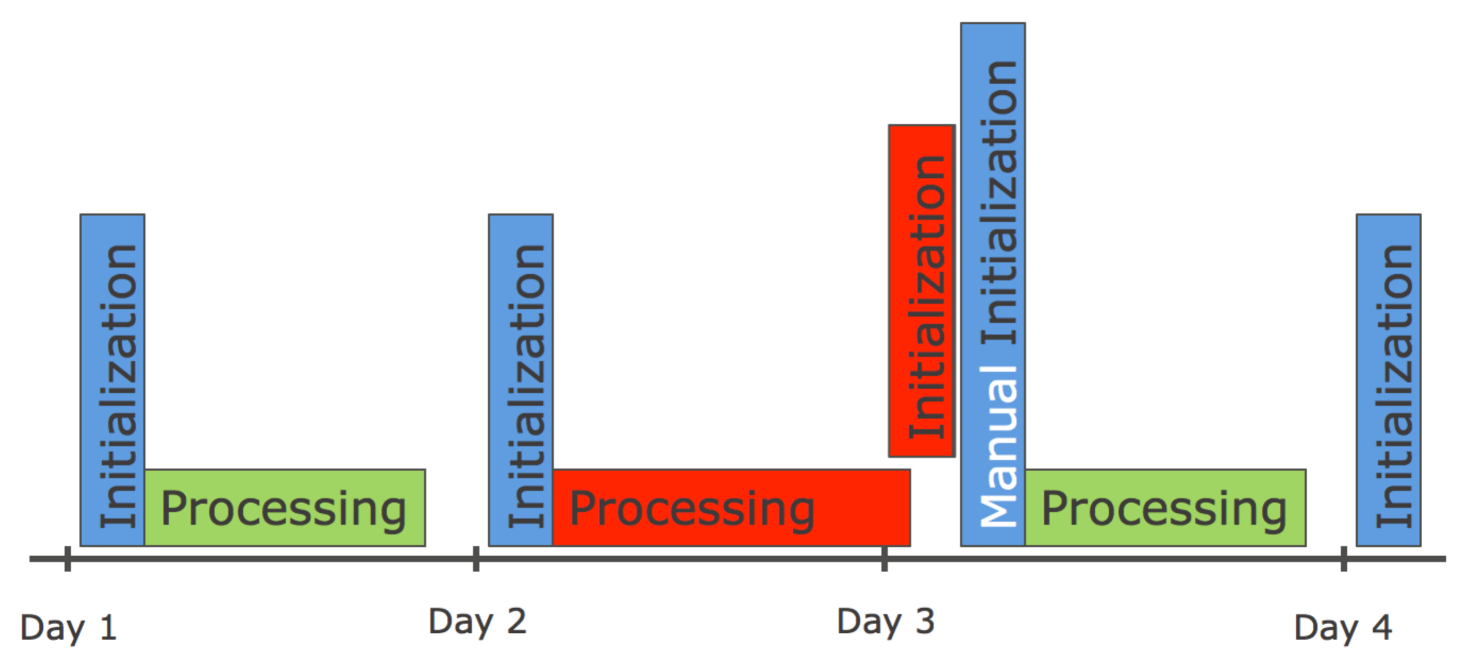


Figure 7 - daily based initialization

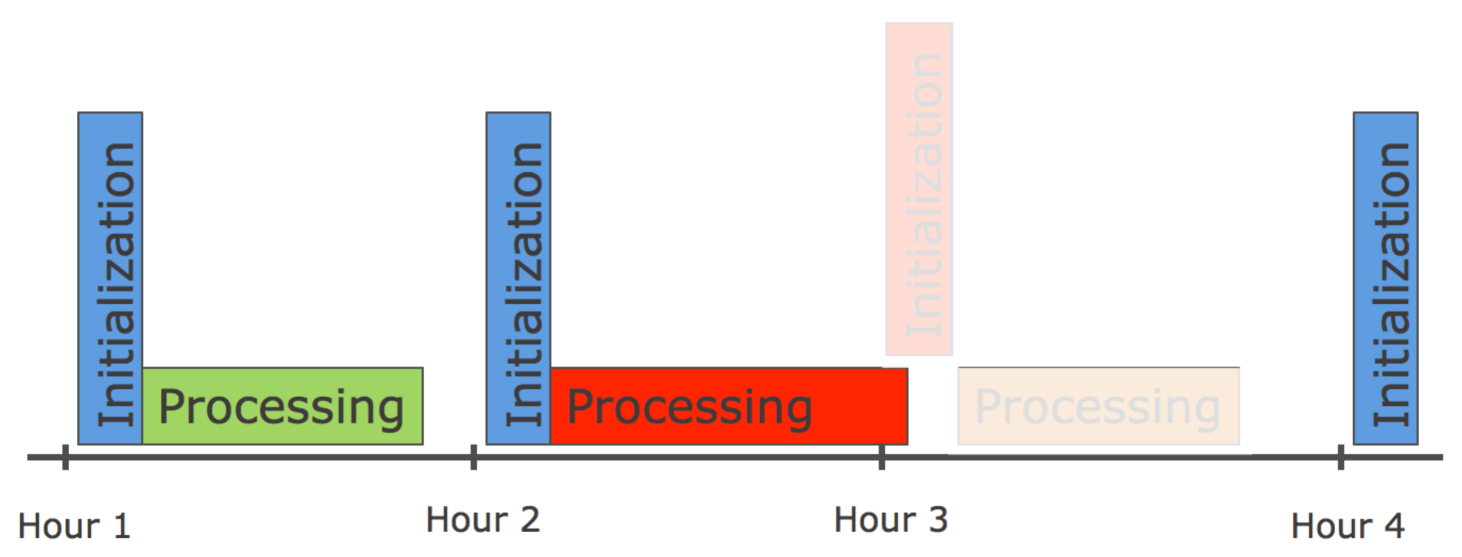


Figure 8 - intra-day initialization

Two pictures above discuss daily and inter-day initialization difference. Every initialization is process, which is scheduled in OS scheduler. Initialization process itself has integrated check for testing if all jobs from the last processing are already finished. If not, initialization cannot continue.

On the first picture the situation in daily load is drawn. Initialization of Day-3 cannot be done due running jobs from Day-2. It is necessary to correct the situation by manual launching of initialization when old day processing is finished. On the second picture the inter-day processing is shown. There is no chance for manual launch of Initialization when the last period jobs are running in time of initialization. In this case Hour-3 jobs are not initialized, but initialization time is updated. Factually it means that there is no Hour-3 processing, Hour-4 processing will have to process the source data together with its data.

The initialization behavior is defined in CTRL\_PARAMETERS table by “INITIALIZATION\_MUST\_RUN” value. For daily processing value 1 guarantee that initialization process will not be skipped, otherwise value 0 for inter-day processing allows unbreakable run in case when jobs are still running in initialization time -> it only updates LOAD\_DATE value. LOAD\_DATE value of running jobs remains the old one.

## Small Initialization

All functionality of small initialization is in procedure SP\_INIT\_PREPARE which is called form PERL script periodically from CRON on weekly, daily or hourly bases. Procedure provides these steps:

1. Check if all jobs for engine\_id are already finished. If not, initialization is skipped therefore operational metadata should be damaged in case when initialization will continue. If INITIALIZATION\_MUST\_RUN=0 some initialization steps are provided, such as new LOAD\_DATE calculation. If INITIALIZATION\_MUST\_RUN=1 no steps are provided, initialization has to be run manually once more when all jobs are finished.
2. CURRENT\_TIMESTAMP of initialization begin is stored into INITIALIZATION\_BEGIN parameter in CTRL\_PARAMETER table.
3. All jobs for engine\_id are deleted from SESS\_JOB table
4. All records for not existing child or parent jobs in SESS\_JOB table are deleted from SESS\_JOB\_DEPENDENCY table
5. If value of INITIALIZATION\_IS\_RUNNING parameter (in CTRL\_PARAMETER table) equal zero, value one is set and next steps are executed, otherwise value of parameter is left and next steps are skipped (value of parameter = 1 means that the last initialization wasn’t finished successfully)
   1. Value of LOAD\_DATE parameter is stored in PREV\_LOAD\_DATE
   2. New LOAD\_DATE calculation
6. Value of APPLICATION\_ID parameter for engine\_id changed to be one (initialization)
7. Scheduler is switched off by setting value of MAX\_CONCURRENT\_JOBS to zero for engine\_id.
8. SESS\_JOB\_DEPENDENCY table is populated by dependency records of small initialization jobs
9. Jobs INITIALIZATION\_STREAM\_BEGIN and INITIALIZATION\_STREAM\_END are created in SESS\_JOB table.
10. All jobs from CTRL\_JOB table having phase = ‘INITIALIZATION’ are transferred into SESS\_JOB table
11. Scheduler is switched on by changing value of parameter MAX\_CONCURRENT\_JOBS. The minimum value from MAX\_CONCURRENT\_JOB\_SET and MAX\_CONCURRENT\_JOBS\_DFLT parameter is taken.

Next Engine starts small initialization jobs which causes that big initialization is provided.

### New LOAD\_DATE calculation

There are two ways how LOAD\_DATE value is calculated based on INITIALIZATION\_CURRDATE\_RELATED

parameter:

* 1. If parameter equals to 0:

new LOAD\_DATE value = current LOAD\_DATE + INITIALIZATION\_RETENTION\_PERIOD

* 1. If paremer equals to 1:

new LOAD\_DATE value = current\_datetime rounded to nearest INITIALIZATION\_RETENTION\_PERIOD [min]. By default value is round down, but it could be rounded up if INITIALIZATION\_CURRDATE\_RELATED\_ROUND\_UP is set to 1.

**Example:** current\_datetime = 2017-10-30 23:59:55, INITIALIZATION\_RETENTION\_PERIOD=5, INITIALIZATION\_CURRDATE\_RELATED\_ROUND\_UP is not set -> new LOAD\_DATE = 2017-10-30 23:55:00, if INITIALIZATION\_CURRDATE\_RELATED\_ROUND\_UP would be set -> new LOAD\_DATE = 2017-11-01 00:00:00.

## Big Initialization

Big initialization is constitutes by small initialization jobs. The small initialization jobs can have different meaning, but typically they have to do:

* Initialization of all processing jobs
* Job’s statistics update
* Other special work execution like logs archiving or tables from UTILITY database archiving
* Initialization end confirmation

#### Job’s initialization

Initialization has to initialize all jobs for engine\_id from CTRL\_JOB table which are not small initialization (phase <> ‘INITIALIZATION’). These jobs can’t be placed into SESS\_JOB table directly therefore Engine is running so all jobs has to be placed into SESS\_JOB\_BCKP table first. All functionality is written in SP\_INIT\_INITIALIZE procedure which is called from PERL script. Procedure provides these steps:

1. Distinct list of runplan values is taken from CTRL\_STREAM\_PLAN\_REF table and list is placed into TEMP\_PLAN table.
2. Value of initial job’s status is calculated for every record in TEMP\_PLAN\_TABLE.
3. Jobs STREAM\_BEGIN and STREAM\_END are created in SESS\_JOB\_BCKP table for all stream from CTRL\_STREAM table which is initialized
4. All regular jobs which is initialized from CTRL\_JOB table are placed into SESS\_JOB\_BCKP table
5. Initial value of job’s status is taken from TEPM\_PLAN table
6. All dependencies are built in SESS\_JOB\_DEPENDENCY\_BCKP table
7. Also dependency of all jobs on INITIALIZATION\_STREAM\_END job which blocked launching jobs before all small initialization jobs are finished is placed
8. Job’s initial status is modified by status\_begin value for all jobs with not null value of status\_begin
9. All records from SESS\_JOB\_DEPENDENCY\_BCKP table is placed into SESS\_JOB\_DEPENDENCY table
10. Finally all records from SESS\_JOB\_BCKP table is placed into SESS\_JOB table

#### Job’s statistics update

Job’s statistics must be updated for possibility to compare current job’s behavior with standard one. Functionality is written in SP\_INIT\_RECALC\_STATISTICS procedure that has these steps:

1. Records from SESS\_STATUS table are used for update of SESS\_JOB\_STATISTICS table:
   1. Avg\_duration value is calculated as difference in seconds between last\_start\_ts and end\_ts for successfully finished jobs
   2. Avg\_end\_tm value is calculated as difference in seconds between end\_ts and timestamp when initialization has finished (INITIALIZATION\_END in CTRL\_PARAMETERS table)
2. Records from SESS\_JOB\_STATISTICS are inserted into STAT\_JOB\_STATISTICS, already retired records are deleted
3. Content of SESS\_STATUS table is backuped into STAT\_STATUS table and deleted
4. Estimates are placed into SESS\_JOB\_STATISTICS table:
   1. Value of day\_in\_week is and day\_in\_month is taken from PREV\_LOAD\_DATE parameter, Monday = 1, Sunday = 7, Ultimo of the month = 999
   2. Rules used for avg\_duration and avg\_end\_tm calculation:
      1. Values for day\_in\_week and day\_in\_month are calculated separately
      2. Only last 15 values are taken
      3. Average is calculated from taken values
      4. The higher value of day\_in\_week and day\_in\_month is taken

#### Initialization end confirmation

Job initialization end confirmation has to be the last initialization job. Values of some parameters in CTRL\_PARAMETERS table are changed. Oracle procedure SP\_INIT\_INITIALIZATION\_END is called from PERL script Init\_Initialization\_end.pl. These parameter’s values are changed by procedure:

1. APPLICATION\_ID is changed to 0 (regular processing).
2. INITIALIZATION\_END is populated by current timestamp
3. INITIALIZATION\_IS\_RUNNING is set to 0

# Framework checker

Framework\_checker is independent application that monitors work of Engine, Scheduler and also current state of all jobs. Application logic is saved in these procedures:

* SP\_FWRK\_CHECK\_WD\_STATUS – Engine check
* SP\_FRWK\_CHECK\_SCHED\_STATUS – Scheduler check
* SP\_FWRK\_CHECK\_NOTIFICATION – job state check
* SP\_FWRK\_CHECK\_INITIALIZATION – initialization check
* SP\_FWRK\_MESSAGE\_GEN – evaluation of events from STAT\_LOG\_EVENT\_HIST table and message generate into STAT\_LOG\_MESSAGE\_HIST table.

## Checker Script

PERL script Framework\_checker.pl is started from CRON and calls all procedures. Procedure checks events and inserts them into STAT\_LOG\_EVENT\_HIST table. The last procedure SP\_FWRK\_MESSAGE\_GEN compares events with alert compartment and creates messages into STAT\_LOG\_MESSAGE\_HIST table. All messages are dispatched as SNMP trap.

## Metadata repository procedures

### Procedure SP\_FWRK\_CHECK\_WD\_STATUS

Procedure is used for correct Engine’s run detection. It calculates difference between last Engine cycle and current timestamp. If value exceeds WATCHDOG\_INTERVAL\_FRWK parameter value, event is created.

### Procedure SP\_FWRK\_CHECK\_SCHED\_STATUS

Procedure checks the Scheduler. Event is created when Scheduler is stopped when not all jobs are already finished.

### Procedure SP\_FWRK\_CHECK\_NOTIFICATION

All jobs which is checked are located in CTRL\_NOTIFICATION table where record is inserted using trigger during inserting job into CTRL\_JOB table. Default value, which is checked is average running time of job. If other notification type is required, manual edit of table is necessary. Column values are set using these rules:

* Job\_name – same as in CTRL\_JOB
* Notification\_enabled – value of NOTIFICATION\_ENABLED\_DFLT parameter
* Notification\_code – CTRL\_NOTIFICATION\_TYPES.notification\_type\_cd WHERE UPPER(notification\_type\_ds) = ‘OPERATOR’
* Avg\_duration\_tolerance – when value of NOTIFICATION\_DFLT parameter equal ‘AVG\_DURATION’, then AVG\_DURATION\_TOLERANCE parameter value, otherwise NULL
* Avg\_end\_tm\_tolerance – when value of NOTIFICATION\_DFLT parameter equal ‘AVG\_END\_TM’, then AVG\_END\_TM\_TOLERANCE parameter value, otherwise NULL
* Checked\_status – NULL
* Max\_n\_run – NULL
* Error\_cd –NOTIFICATION\_DFLT parameter value

If several events for one job is required, it is necessary create new record for every event which will be checked. The reason for that is uniqueness of error\_code for every event; therefore it is used for recommendation choice.

### Procedure SP\_FWRK\_CHECK\_INITIALIZATION

Procedure checks a correctness of initialization by checking if:

LOAD\_DATE + INITIALIZATION\_DELAY\_DAYS\*24 + INITIALIZATION\_RETENTION\_PERIOD\*1440 + INITIALIZATION\_HOUR is greater than CURRENT\_TIMESTAMP when INITIALIZATION\_HOUR >= 0. If not, event is created. Also duration of initialization is checked, when duration exceed INITIALIZATION\_DURATION\_MINUTES parameter vale, event is created

### Procedure SP\_FWRK\_MESSAGE\_GEN

Procedure checks new records in STAT\_LOG\_EVENT\_HIST table and insert new message into STAT\_LOG\_MESSAGE\_HIST table:

* If event is still active, new message is created when MESSAGE\_REITERATION\_INTERVAL parameter value is missed and MESSAGE\_REITERATION\_COUNT parameters value is not exceeded.
* If Engine event appears, the message is generated for every event
* Message Scheduler alert is created if number of Scheduler event is actual and appears SCHEDULER\_OFF\_COUNT parameter value times during last hour
* Message Initialization alert is created if number of Initialization event is actual and INITIALIZATION\_PROBLEM\_COUNT parameter value times during last hour

Value transformation between STAT\_LOG\_EVENT\_HIST.notification\_cd and STAT\_LOG\_MESSAGE\_HIST .notification\_type\_cd using transformation from CTRL\_NOTIFICATION\_RLTD lookup sometimes causes records multiply. It is necessary for respecting channel possibility.

# GUI application

GUI application is WEB based application supporting standard browser functionality and using Teradata Viewpoint graphical layout. The user must have appropriate permissions for working with any part of the application. The main ask on GUI application is to enable comfortable monitoring and controlling of job processing, but GUI application is not necessary for PDC work. Jobs can be located in different instances of PDC; these instances are totally independent and are used for controlling and monitoring independent data processing. GUI shows only jobs located in selected instance of PDC but on status line are displayed statuses of all instances simultaneously. For selecting only specific part of objects a filter can be used on stream and job level. GUI consists from several parts whose meaning and functionality is described below.

## Logon Page

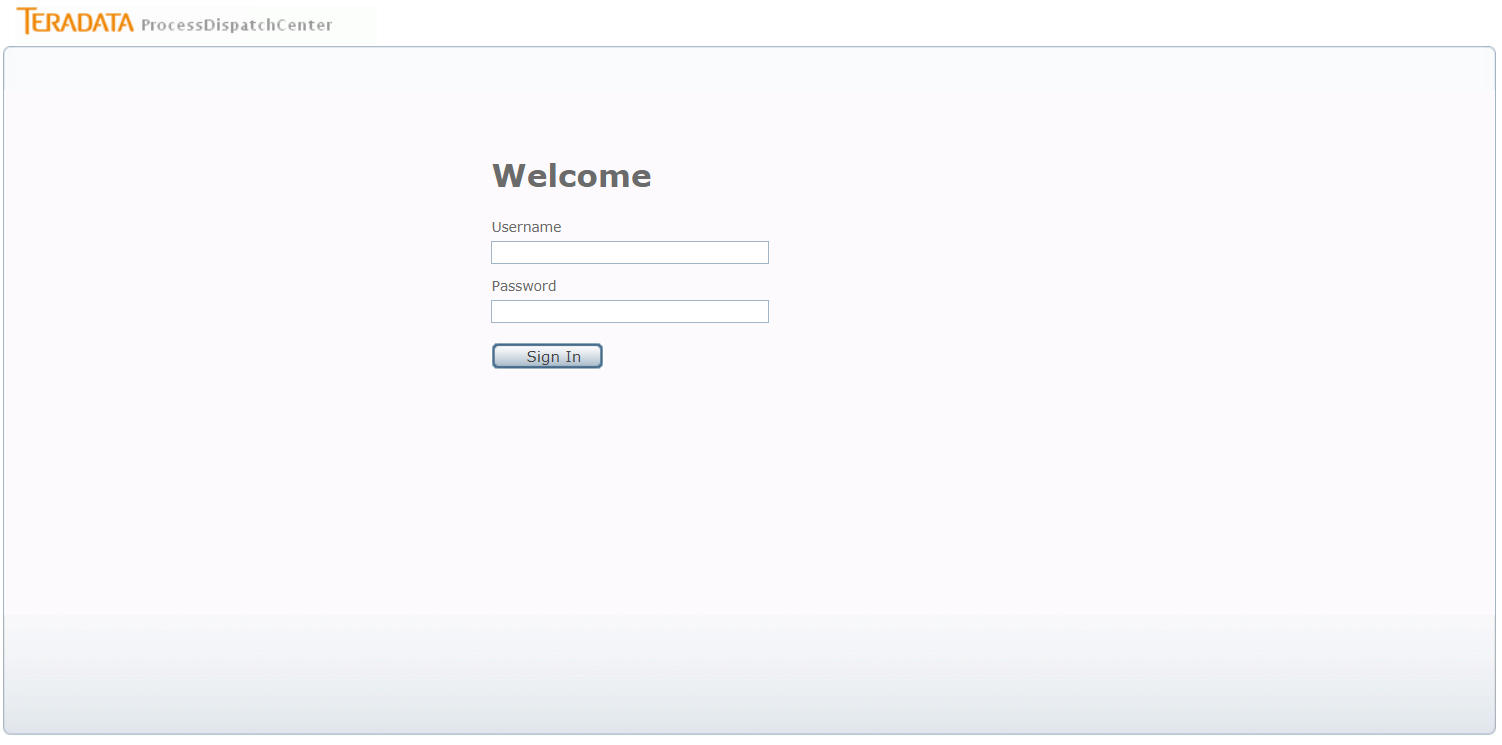


Figure 9 - Logon page

Picture shows the logon page layout. All information in PDC application is shown only when user have adequate rights to see or control it.

## Page Layout

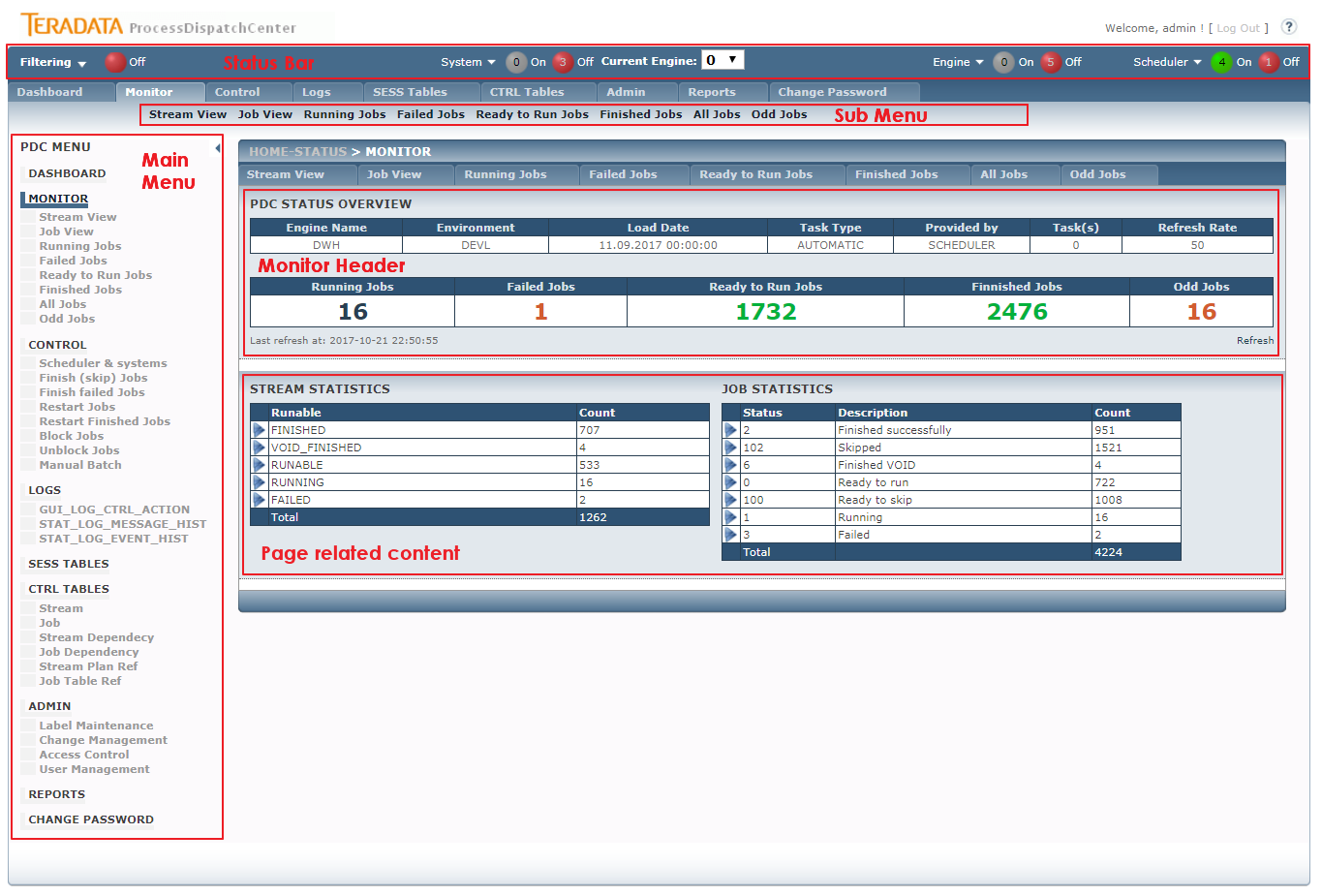


Figure 10 - GUI page layout

Picture shows the main parties of PDC application.

The Status Bar is located on the top of page. It is used for displaying Engine’s and Scheduler’s statuses and entering to filter menu for information filtering. All displayed data is related to Engine number, which is selected in the middle of Status Bar.

Main Menu is located on the left side of the page and secondarily also under the Status Bar. Left menu can be used for direct access into menu and submenu entry. This menu can be hidden if more places for Page Related Content is necessary.

Submenu is used in case that main menu is hidden for submenu entry access.

Monitor header is the most important part of PDC application. There are shown all processing related information about jobs. More detail will be discussed later.

“Page Related Content” part shows chosen information that depends on menu entry, which is selected.

*Note: The default “blue” color schema can be changed for better recognition environment. Modify GUI\_COLOR parameter in CTRL\_PARAMETERS table with mean-full color name or with color hexadecimal value.*

### Status Bar

#### Filter Menu

The Filter Menu is located in left part of Status Bar. The filter works as global filter on all values related filter entry. When filter is active, only records satisfied filter condition are displayed. The search value is case sensitive and you can choose if it will be processed with or without % wildcard (e.g. “LIKE” %value%).

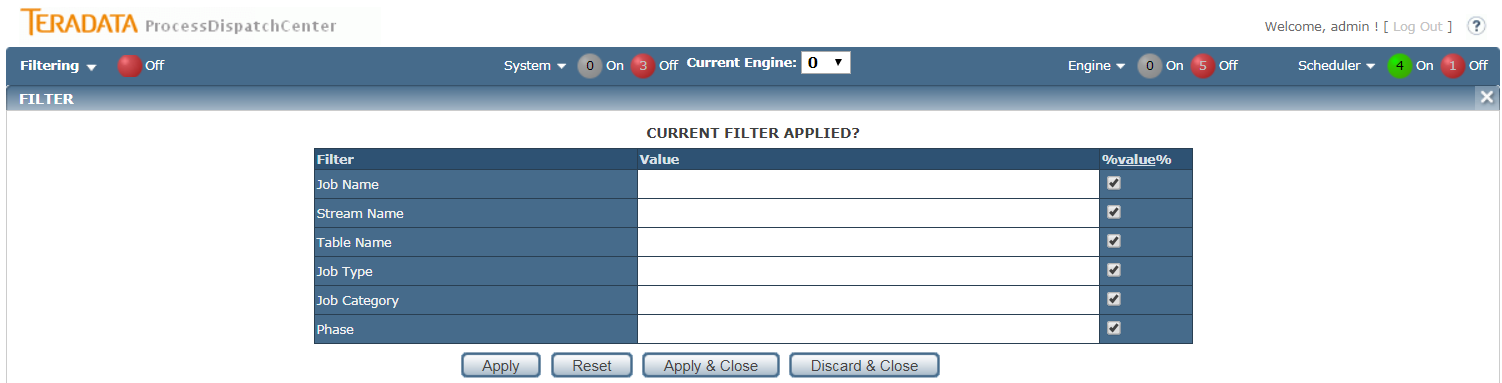


Figure 11 - GUI filtering box

#### Current Engine

There is no limitation on Engine count. Generally is supposed that Engine ID = 0 is used for regular batch processing. Therefore no Engine has any dependency on other Engine, PDC application shows only information related to Engine ID which is selected.

#### System

In System part the numbers of running and off systems are shown. If you click on system, systems’ overview is shown.

#### Engine

In Engine part the numbers of running and off engines are shonw. If you click on Engine, engines’ overview is shown.

#### Scheduler

In Scheduler part the numbers of running and off Scheduler are shown. If you click on Scheduler, shedulers’ overview is shown.

### Main Menu

Menu is divided into main section such as Monitoring, Control and so on. Most section contains submenu entry. Meaning of each menu entry will be discussed later. The menu content can be hidden if more places for Page Related Contend is necessary. When main menu is hidden you can still touch entry in Menu and Submenu part on top part of the page.

### Monitor Header

Monitor header shows the most important information about batch processing. This information is displayed:

* Environment – the environment kind which is monitored such as PROD, TEST or DEVL. There is no limitation on number or kind of environments but please note that every environment needs separate instance of metadata database and GUI application. Shown value is taken from CTRL\_PARAMETERS table.
* Load Date – load date or load timestamp for intermediate load is displayed
* Task Type – AUTOMATIC for regular batch processing is shown. MANUAL is displayed when Manual batch is in progress.
* Provided by – in case of Manual batch processing the user name that initialized Manual batch is shown, otherwise SCHEDULER is displayed.
* Task(s) – maximal number of running jobs concurrently is shown.
* Refresh Rate – refresh rate in second of information placed on Monitor Header is displayed
* Running – number of currently running jobs is displayed. Number works as hyperlink to corresponding page which shows job’s details of running jobs.
* Failed Jobs – number of failed jobs is displayed. Please note that it’s big difference between Failed jobs and jobs in failed status. There are only placed jobs which already reach maximal number of automatic restart. These jobs have to be restarted by supervisor. Number is hyperlink to corresponding page which shows job’s details.
* Ready to Run Jobs – number of jobs not launched yet. Number is hyperlink to corresponding page which shows job’s details.
* Finished Jobs – number already finished jobs. Number is hyperlink to corresponding page which shows job’s details.
* Odd Jobs – number of jobs running oddly is displayed. The main reason for job become oddly is that job is running out of statistics. Number is hyperlink to corresponding page which shows job’s details.

Tip: The best guidance for supervising is holding values in:

* Running – the value grater then zero means that some jobs are currently running, zero value, when job’s processing is not finished yet, is unwanted.
* Failed Jobs – when not zero value is displayed, the job’s problem has to be solved
* Odd Jobs – not zero value says that some jobs are running out of statistics, the potential job’s freezing has to be revised.

### Status line

Status line is located in the top level of GUI application and shows the number of the engine which is selected for monitoring and controlling. Statuses of other Engines and Schedulers are also displayed. For faster touching of information which is looked for a filter can be used.

## DASHBOARD

Dashboard allows to get overview about all processes handled by engines. Number of running jobs, failed jobs and odd jobs is shown. After click on one of those numbers monitor page of related engines is shown. Progrress bar of processing, load date and scheduler state is also shown for each engine.

You can filter showed enignes by menu showed after click on filtering link.

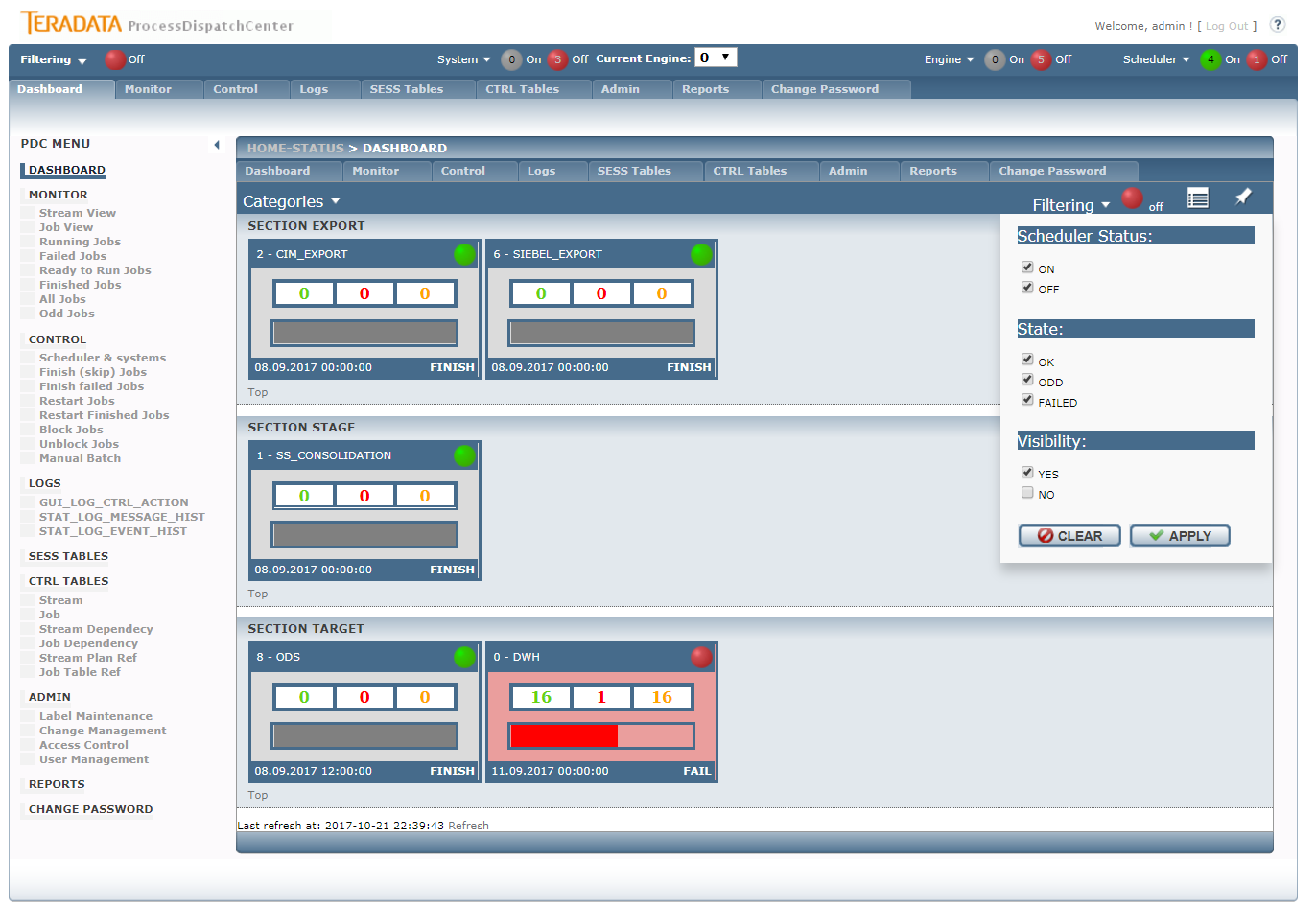


Figure 12 - GUI dashboard box view

There are two pageview modes: boxing view and line view. Mode is change by click on the icon button next to filtering.



Figure 13 - GUI dashboard line view

## Monitor

Monitor is used for displaying of current status of processing. The information can be presented from stream or job point of view. Drill down functionality enables drill for further details, so from stream view the user can drill down the information of how many jobs are located in the stream, what they are and what their status is. User can also drill down to parameters of selected job. On job level the user operates the job; it means he can abort running job, restart or finishing failed job and so on. Jobs and streams are divided into processing classes which represents objects state such as prepare for run, running, finished, failed and so on. The environment, load date and task type is shown as well as display refresh rate and maximal number of concurrently running jobs. The status overview part contains number of currently running jobs, number of failed jobs, number of jobs prepared for run and number of already finished jobs shows. All these numbers support drill down functionality, it means you can directly get a list of jobs in a category by clicking on the appropriate number.

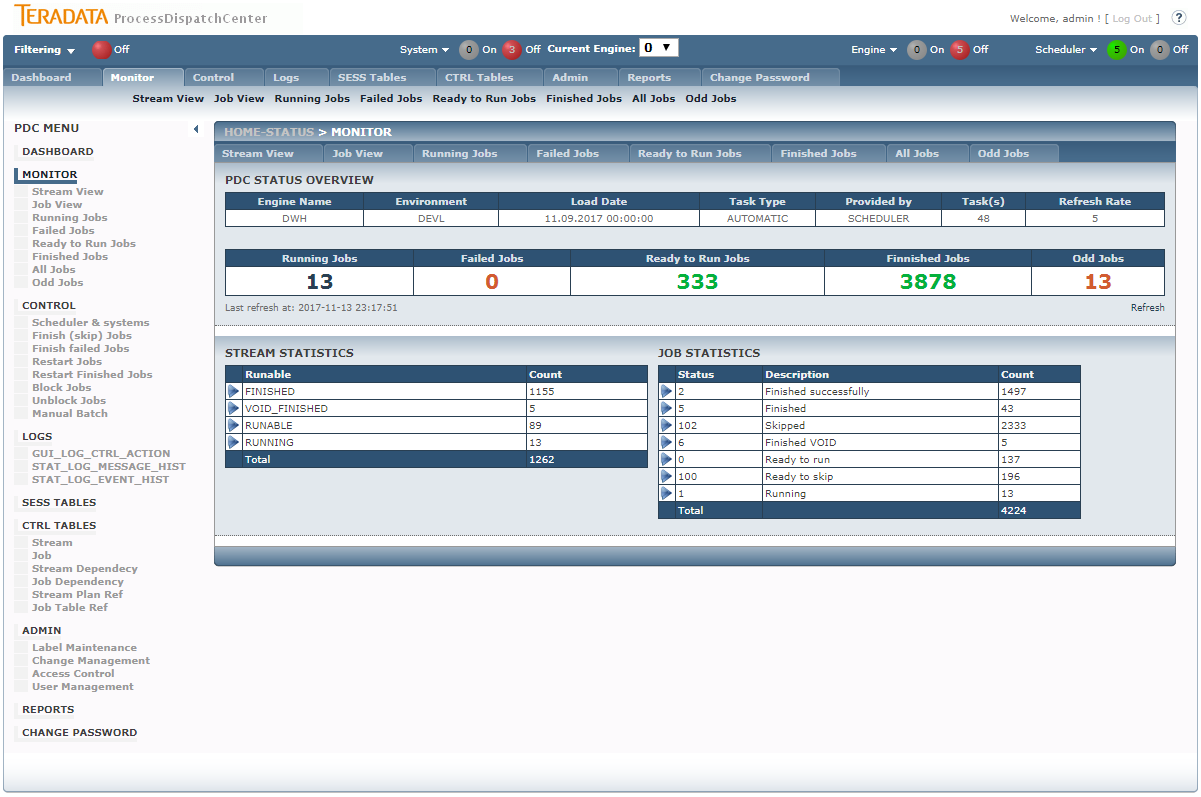


Figure 14 - GUI Monitor page

Picture shows the basic Monitor page. Only Page Related Content will be discussed, all other parts were already discussed above. There are two different lookup possible on processing – the stream related view and job related view. In both parties the number of jobs or streams in defined status is displayed. From stream perspective there are these status categories available:

* FINISHED – all jobs in stream are already finished
* FORCE\_FINISHED – all jobs in stream are already finished, some jobs were force finished (job’s task wasn’t realized)
* VOID\_FINISHED – all jobs in stream are already finished, some jobs failed but successfully finishing of these jobs weren’t required
* FINISHED\_ODDLY - all jobs in stream are already finished, some jobs finished oddly
* RUNABLE – some jobs of stream weren’t launch yet
* RUNNING – some jobs of stream are currently running
* FAILED – some jobs of stream are in failed status
* BLOCKED – some jobs of stream are manually protected to launch

From job view there are many statuses which saying in which state job currently appears. The meaning of statuses is similar like statuses of the stream. Several statuses have same meaning even if number differs therefore status “remember” history of job’s processing.

Monitor pages allows to show excecution logs by click on show log(s) button if job details are shown (click on right arrow in job list overviews).

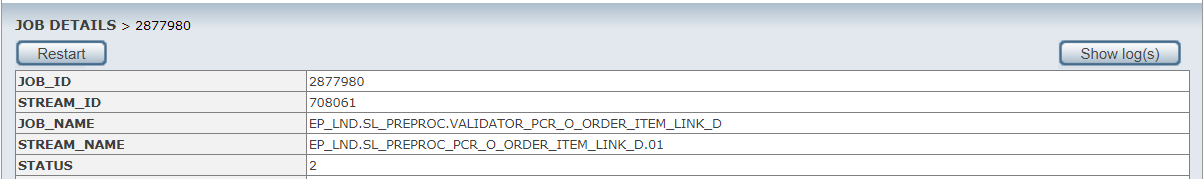


Figure 15 - job details part of monitor page



Figure 16 - log shown via GUI

Log is shown in new pop-up window. You can download log by button on the left side next to close button.

### Monitor – Stream View



Figure 17 - GUI Monitor – Stream view with details

All status categories in stream view are clickable. Clicking on it, the stream list assigned into this category appears. Detail table shows number of jobs in each status category for every stream. Clicking on stream the list of jobs assigned to this stream appears.

### Monitor – Job View

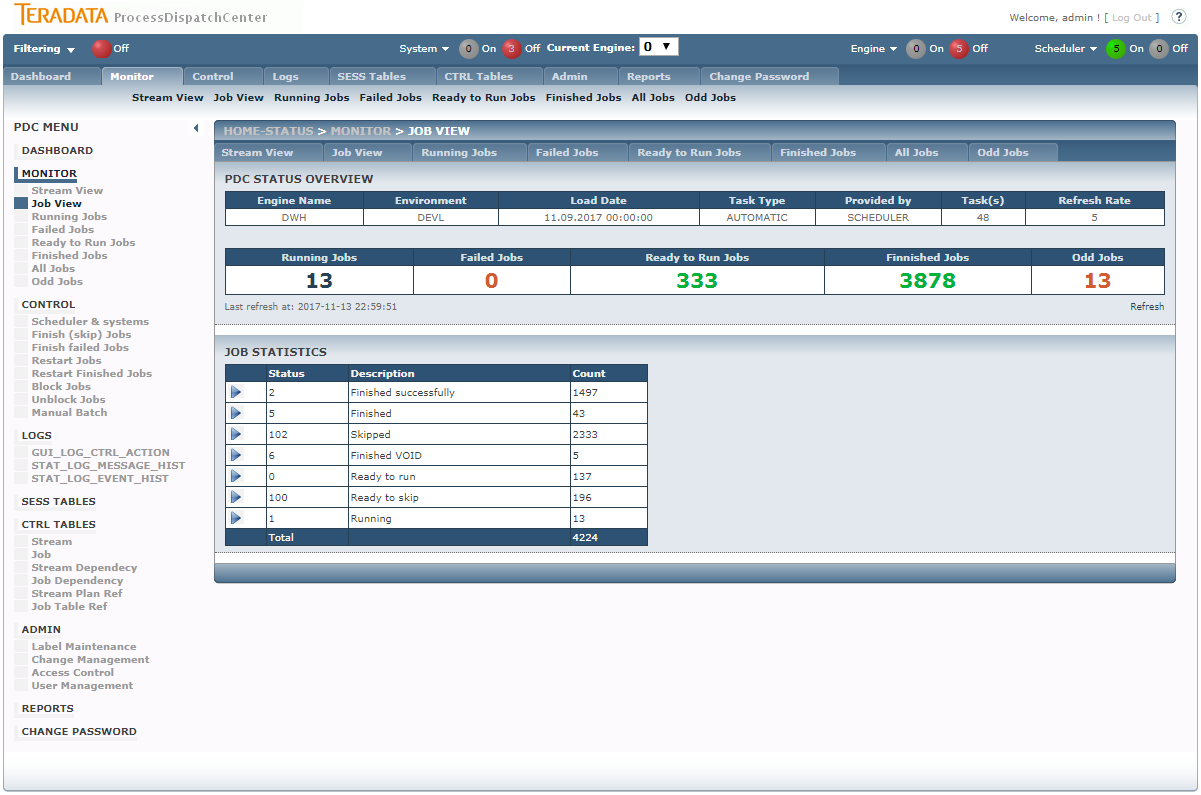


Figure 18 - GUI Monitoring – Job View page

Monitoring – Job View page shows separate statuses and number of jobs currently having this status. All statuses are clickable. When clicking on it, the list of jobs in status appears. Clicking on job displays job’s details. This section is identical with detail’s section of the next monitoring pages and will be discussed in this section.

### Monitor – Running Jobs

There are two possibilities how to reach this page. The first is using Running Jobs entry in menu, the second one is clicking on Number of Running Jobs in Monitoring Header section.

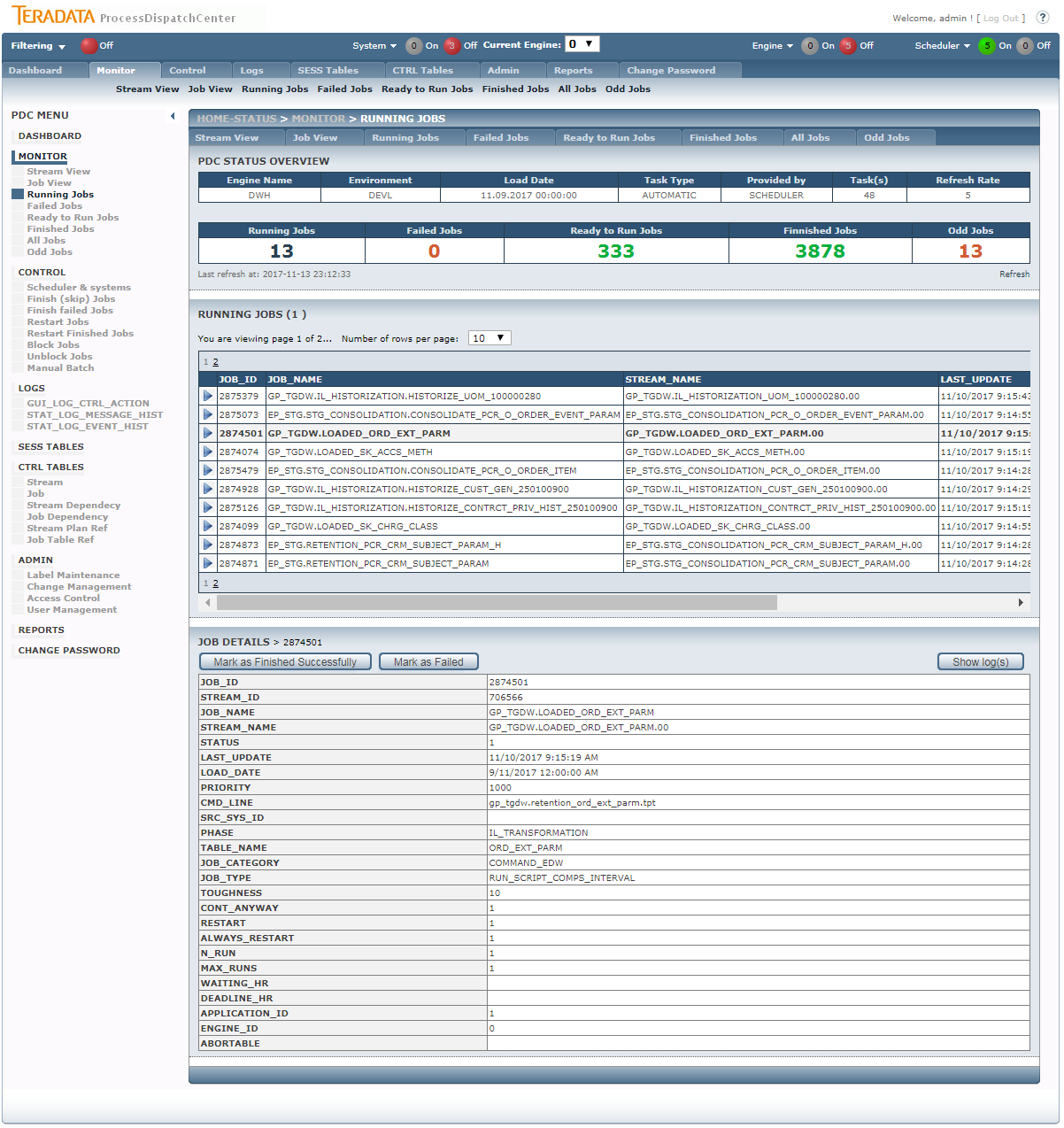


Figure 19 - GUI Monitoring – Running Jobs with details

Monitoring – Running Jobs displays the job list which is currently running. Please note that there is no automatic refresh of job’s details due ensure comfortable work with them, so the list couldn’t be current after some piece of time. Clicking on job the job’s details from SESS\_JOB table appears. On the top part of job’s detail there are buttons for job operation. Buttons have this functionality:

* Abort Job – only when job is abortable is displayed. Clicking on this button, the special job for aborting of running job is started.
* Mark as Finished Successfully – marked jobs as successfully finished. This functionality is used only in case when job finished successfully but job’s metadata from oddly reason aren’t updated.
* Mark as Failed – marked jobs as failed. This functionality is used only in case when job failed but job’s metadata from oddly reason aren’t updated.

### Monitor – Failed Jobs

There are two possibilities how to reach this page. The first is using Failed Jobs entry in menu, the second one is clicking on Number of Failed Jobs in Monitoring Header section.

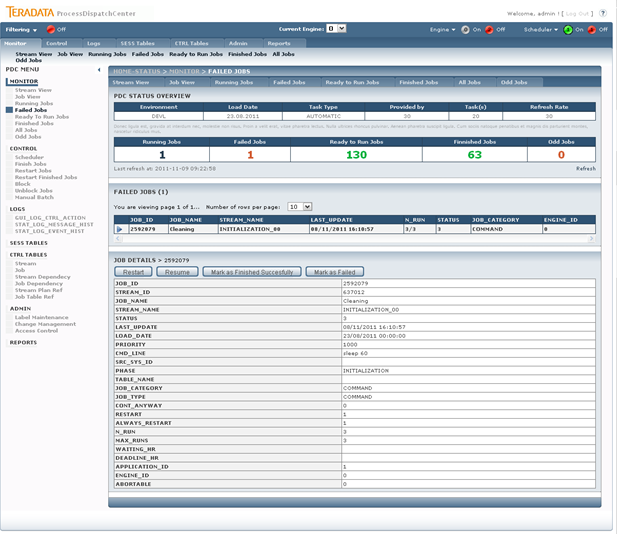


Figure 20 - GUI Monitoring – Failed Jobs page with details

On this page there are only jobs which are in status fail and already reach the number of automatic restart (n\_run = max\_runs) displayed. These jobs have to be operated by supervisor. On the top part of job’s detail page there are buttons for job operation. Buttons have this functionality:

* Restart – job is returned to Scheduler by increasing value of max\_runs parameter by the value of max\_runs parameter from CTRL\_JOB table. The value of restart parameter is set to true even if resume of job is permited.
* Resume – job is returned to Scheduler by increasing value of max\_runs parameter by the value of max\_runs parameter from CTRL\_JOB table. The value of restart parameter is set to false only when value of always\_restart is set to false also. It means that resume has the same functionality as restart for jobs with value of always\_restart parameter set to true.
* Mark as Finished – sending the signal for force finishing job. The job is removed from processing without its successfully ending, child jobs can be launched.

### Monitor – Ready to Run Jobs

There are two possibilities how to reach this page. The first is using Ready to Run Jobs entry in menu, the second one is clicking on Number of Ready to Run Jobs in Monitoring Header section.



Figure 21 - GUI Monitoring – Ready to Run Jobs with details

There are shown jobs prepared for launch on this page. It doesn’t matter if jobs are really prepared, it means having no dependency, no conflict with currently running or failed jobs and so on, or not. On the top part of job’s detail there is button for job operation. Button has this functionality:

* Mark as Finished – this button releases the job from processing. Factually the job can’t be removed therefore ending of the job causes removing its dependency. From this reason only status of the jobs is changed and job is finished by Scheduler when it comes into processing.

### Monitor – Finished Jobs

There are two possibilities how to reach this page. The first is using Finished Jobs entry in menu, the second one is clicking on Number of Finished Jobs in Monitoring Header section.

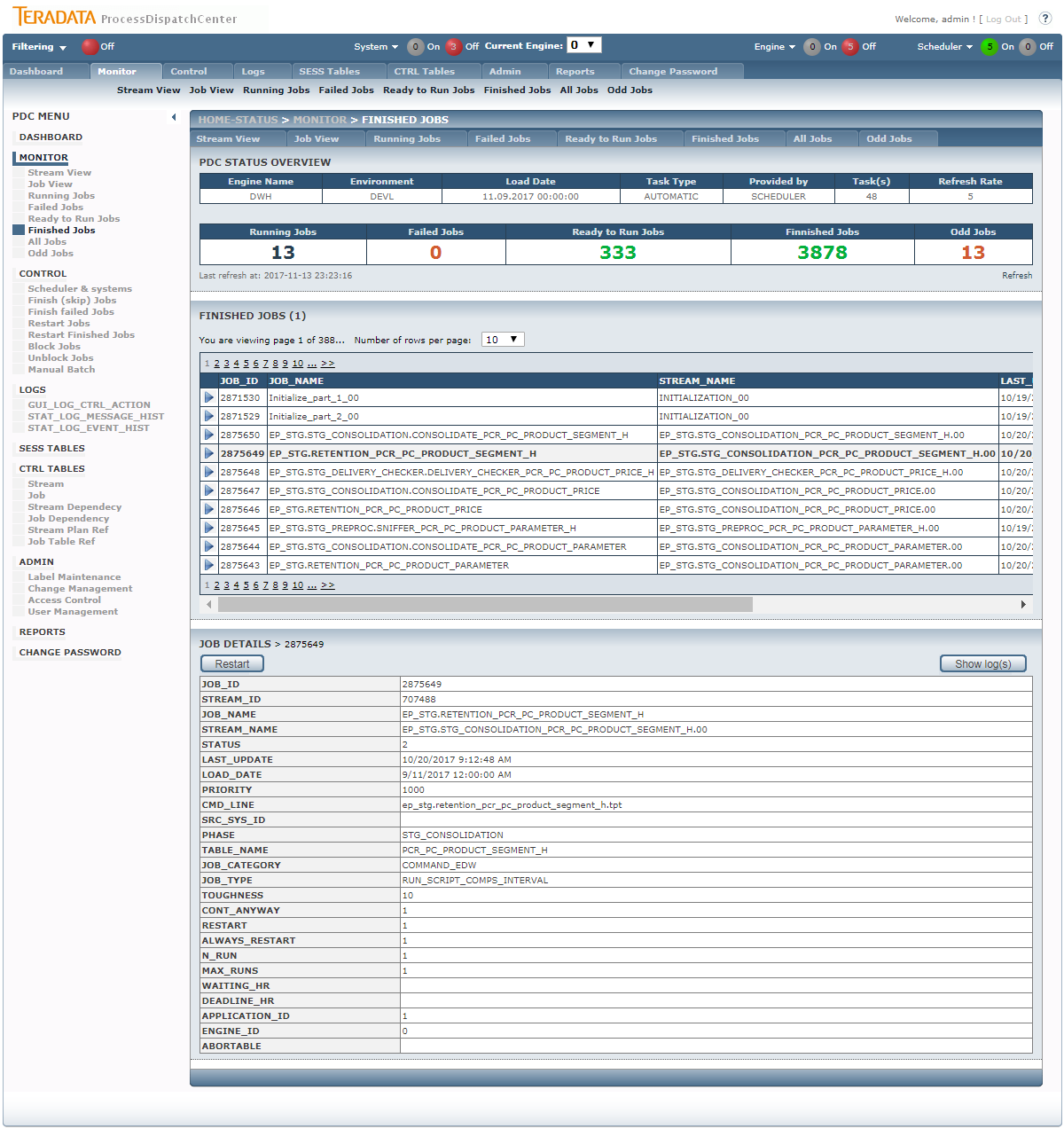


Figure 22 - GUI Monitoring – Finished Jobs with details

There are shown jobs already finished on this page. On the top part of job’s detail there is button for job operation. Button has this functionality:

* Restart – job is returned to Scheduler to be launched again. It is difference between Restart here and Restart in Failed Jobs section. Here already finished job is asked to be restarted, so it is not enough only increasing value of n\_run parameter, but also status of the job has to be changed from finished to ready to run. Please remember that there are no job dependency in time of job restart, so it is not possible simply restart parent and child job together therefore Scheduler doesn’t know anything about execution order of these jobs now. If this task is required, you have to restart parent job first and wait until it finished and then restart child job. For comfortable processing of group of jobs, only Manual Batch is preferred to be used.

### Monitor – All Jobs

This page can be reached only from menu entry therefore there is no special meaning of the jobs listed on this page. This page is used in advance for searching within jobs when you are looking for job’s information and you have no knowledge about current job status.

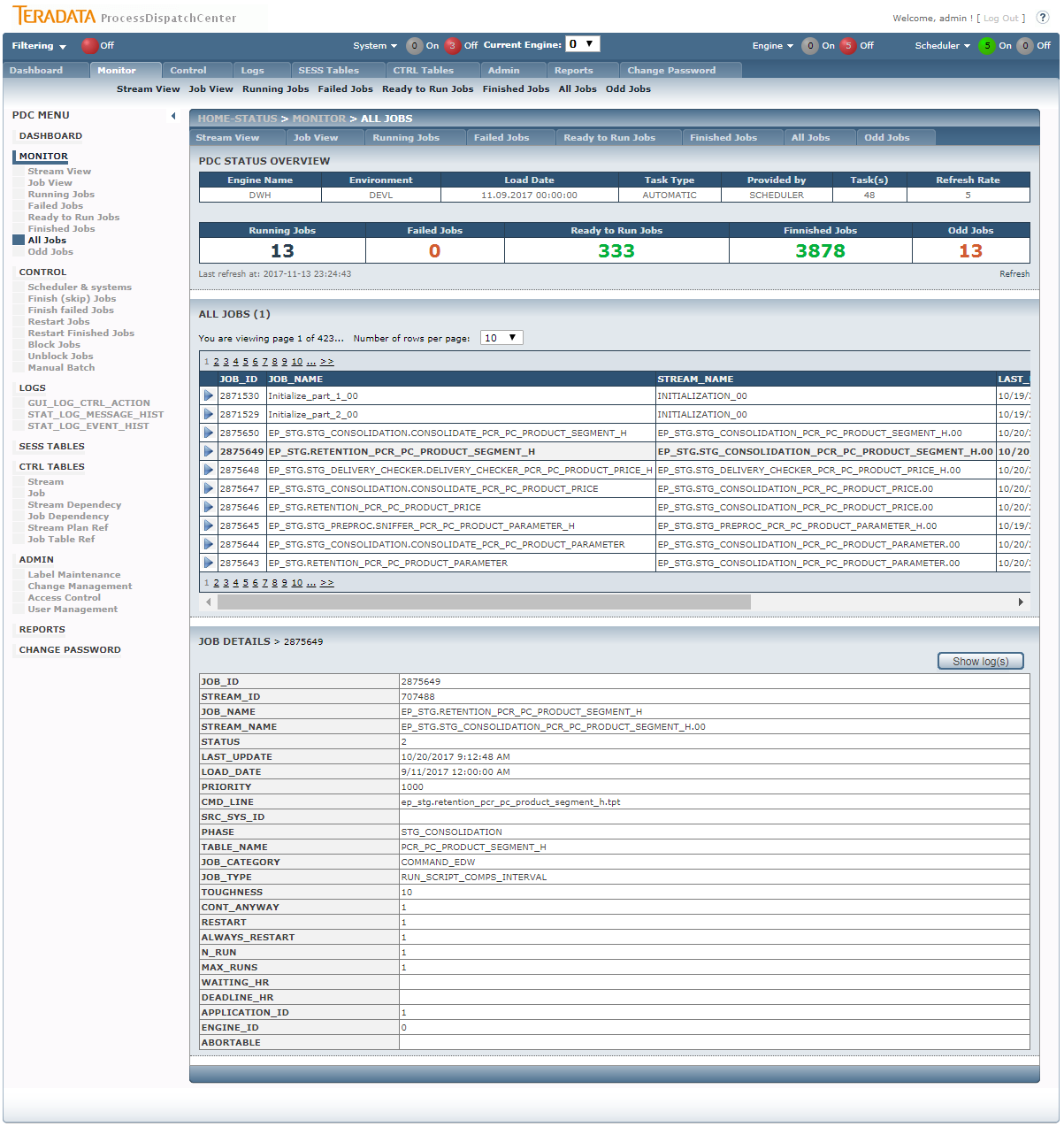


Figure 23 - GUI Monitoring – All Jobs with details

When jobs in different statuses are displayed, there is no button available on this page.

### Monitor – Odd Jobs

There are two possibilities how to reach this page. The first is using Odd Jobs entry in menu, the second one is clicking on Number of Odd Jobs in Monitoring Header section.

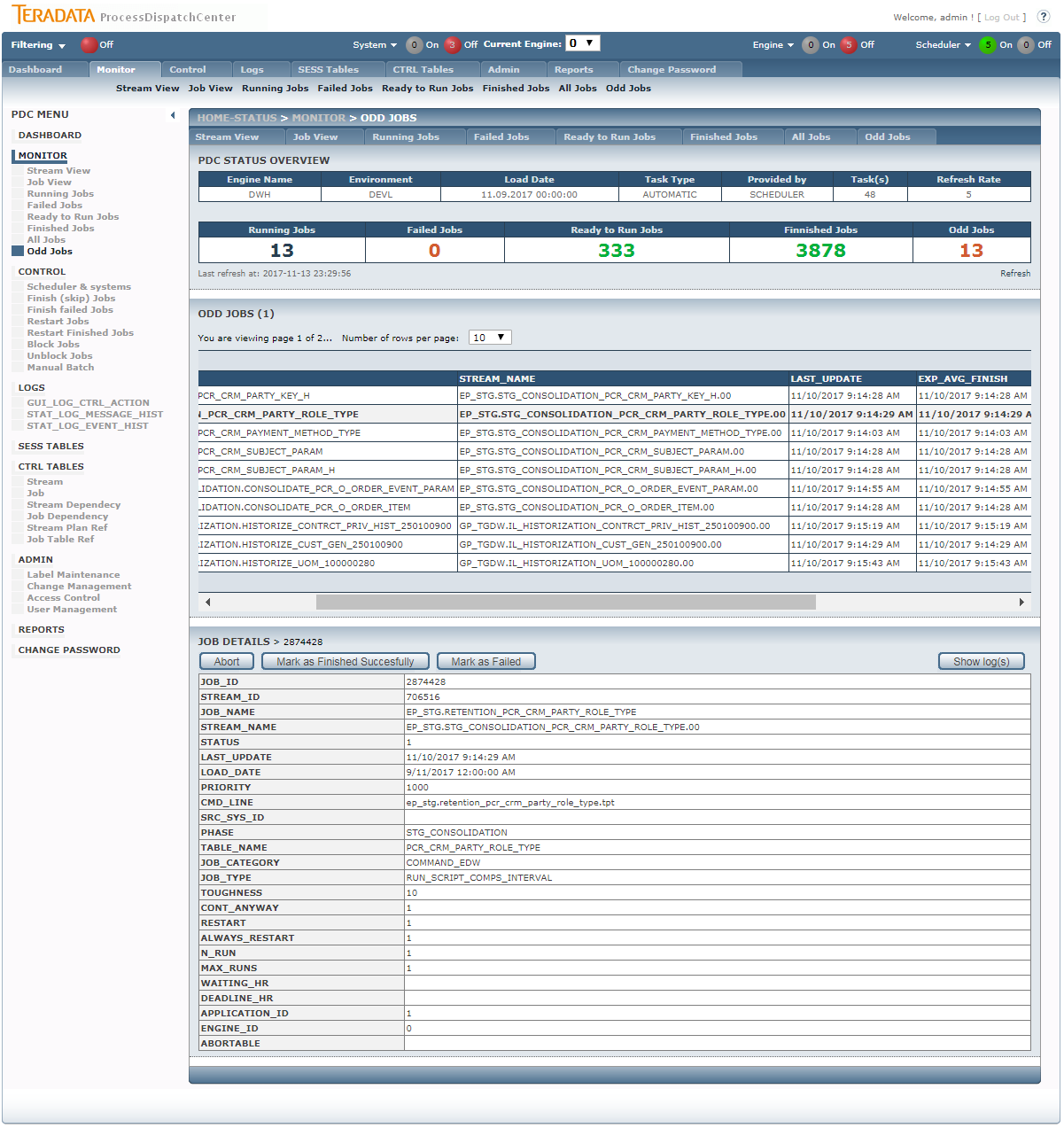


Figure 24 - GUI Monitoring – Odd Jobs with details

There are shown currently running jobs on this page, which average running time, coming from statistics, was exceeded. This situation gives a signal to supervisor for checking the correctness of job processing. On the top part of job’s detail there is button for job operation. Button has this functionality:

* Abort job – only when job is abort-able is displayed. Clicking on this button, the special job, which aborts job, is started.
* Mark as Finished Successfully – marked jobs as successfully finished. This functionality is used only in case when job finished successfully but job’s metadata from oddly reason aren’t updated.
* Mark as Failed – marked jobs as failed. This functionality is used only in case when job failed but job’s metadata from oddly reason aren’t updated.

## Control

Control enables access to all necessary settings used for controlling of PDC application. Same basic functionality for job processing is also located on the monitor part, but Control part enables driving parallelisms, temporary stop the Scheduler or doing control task on job level simultaneously for group of jobs. User can also simply stop executing jobs on selected dependency branch by blocking its parent job. Manual Batch functionality enables recalculation of selected jobs for chosen load date. It’s often used for datamarts recalculation for the day.

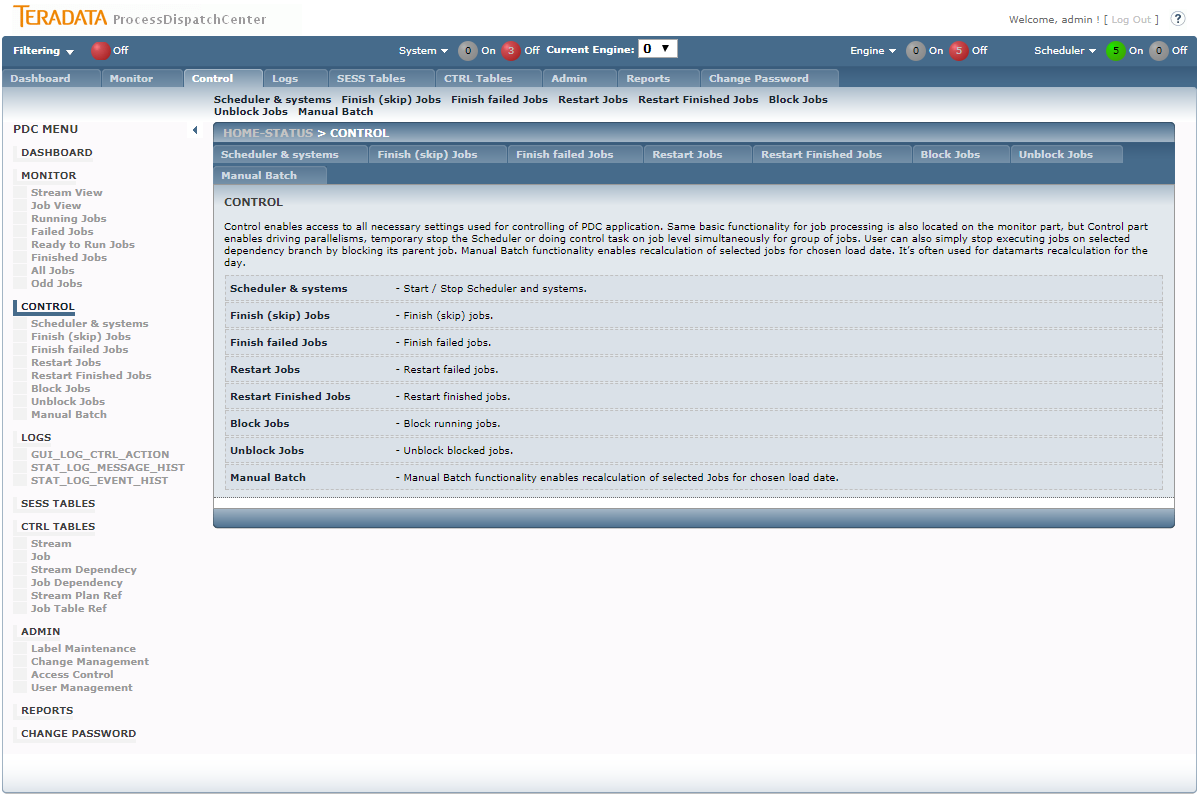


Figure 25 - GUI main Control page

On Control page functionality description is displayed only.

### Control – Scheduler

This page is used for Scheduler control. You can simply stop the Scheduler or change maximal number of currently running jobs for each Scheduler. You can also block system (it’s applied on all engines).

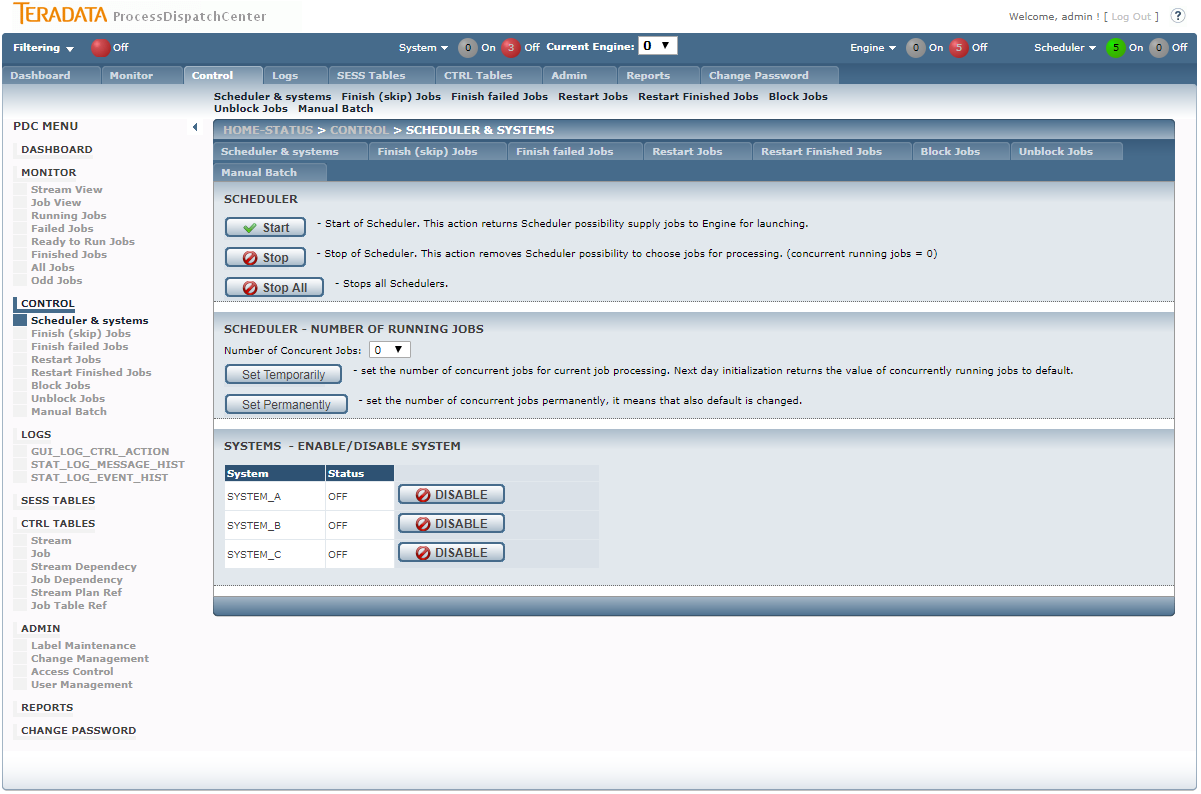


Figure 26 - GUI Control – Scheduler page

There are several buttons for Scheduler control on this page:

* Start – Start of Scheduler. This action returns Scheduler possibility supply jobs to Engine for launching. The engine\_id number of Scheduler is taken from Status Bar value of Current Engine.
* Stop – Stop of Scheduler. This action removes Scheduler possibility to choose jobs for processing. Factually this action set value of number of concurrent running jobs to zero. No action is applied on already running jobs. The engine\_id number of Scheduler is taken from Status Bar value of Current Engine.
* Stop All – stops all Schedulers (for all Engine\_ids).
* Number of Concurrent Jobs – you can set the maximal number of concurrently running jobs. Please remember that there are also limits on job category and subcategory which are real time effective.
* Set Temporarily – set the number of concurrent jobs temporarily, it means for current job processing. Next day initialization returns the value of concurrently running jobs to default. The engine\_id number of Scheduler is taken from Status Bar value of Current Engine.
* Set Permanently - set the number of concurrent jobs permanently, it means that also default is changed. The engine\_id number of Scheduler is taken from Status Bar value of Current Engine.

### Control – Finish failed Jobs

This page is used for ending of failed jobs; in this case it means jobs having status Failed. The functionality is the same as it is when button Mark as Finished on Monitor – Failed Jobs is used, but there are some ads on this page:

* Multiple or all jobs can be selected
* Also jobs with not already consumed limit of restarts can be selected

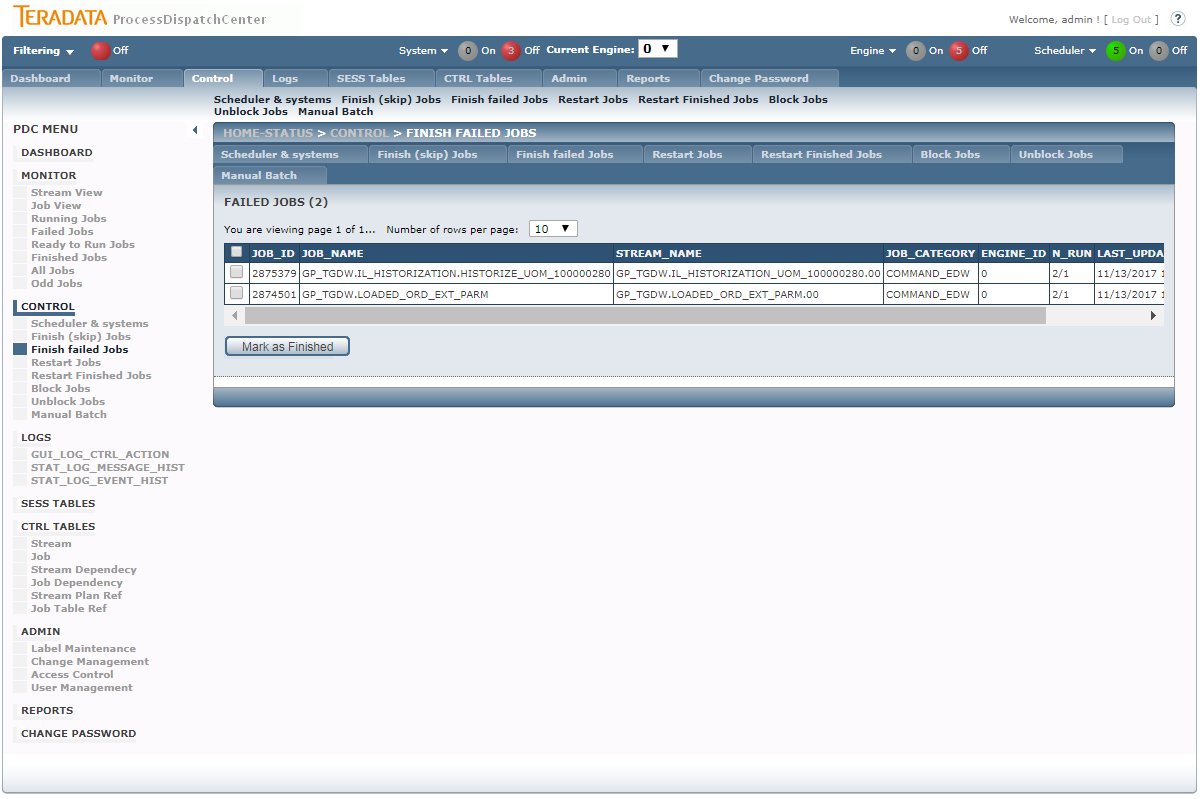


Figure 27 - GUI Control - Finish failed Jobs page

Pressing the Mark as Finished button the signal Finished is send to the jobs which causes ending of job immediately.

### Control – Finish (skip) Jobs

This page is used for skipping jobs which have not finished yet.

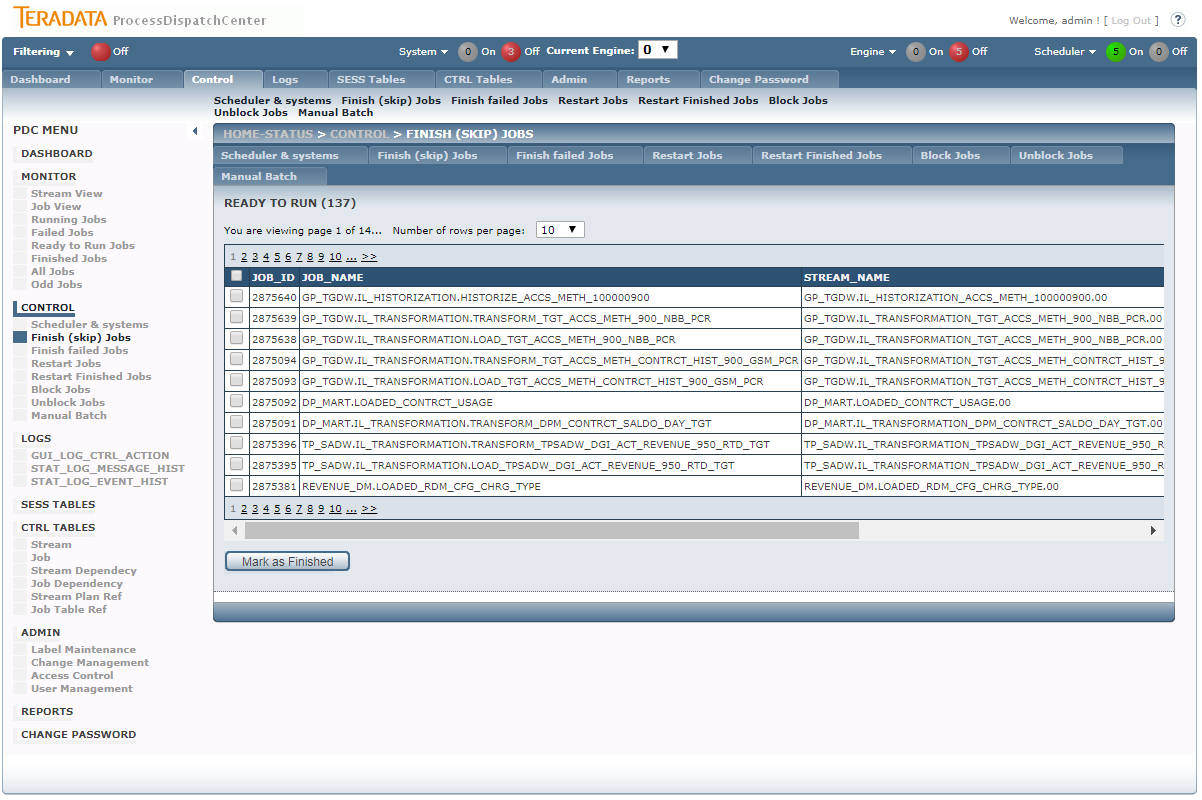


Figure 27 - GUI Control - Finish (skip) Jobs page

Pressing the Mark as Finished button the signal Finished is send to the jobs which causes ending of job immediately.

### Control – Restart Jobs

This page is used for restarting of failed jobs, in this case it means jobs not only having status Failed, but also jobs already consumed all possibility for auto restart by Scheduler.

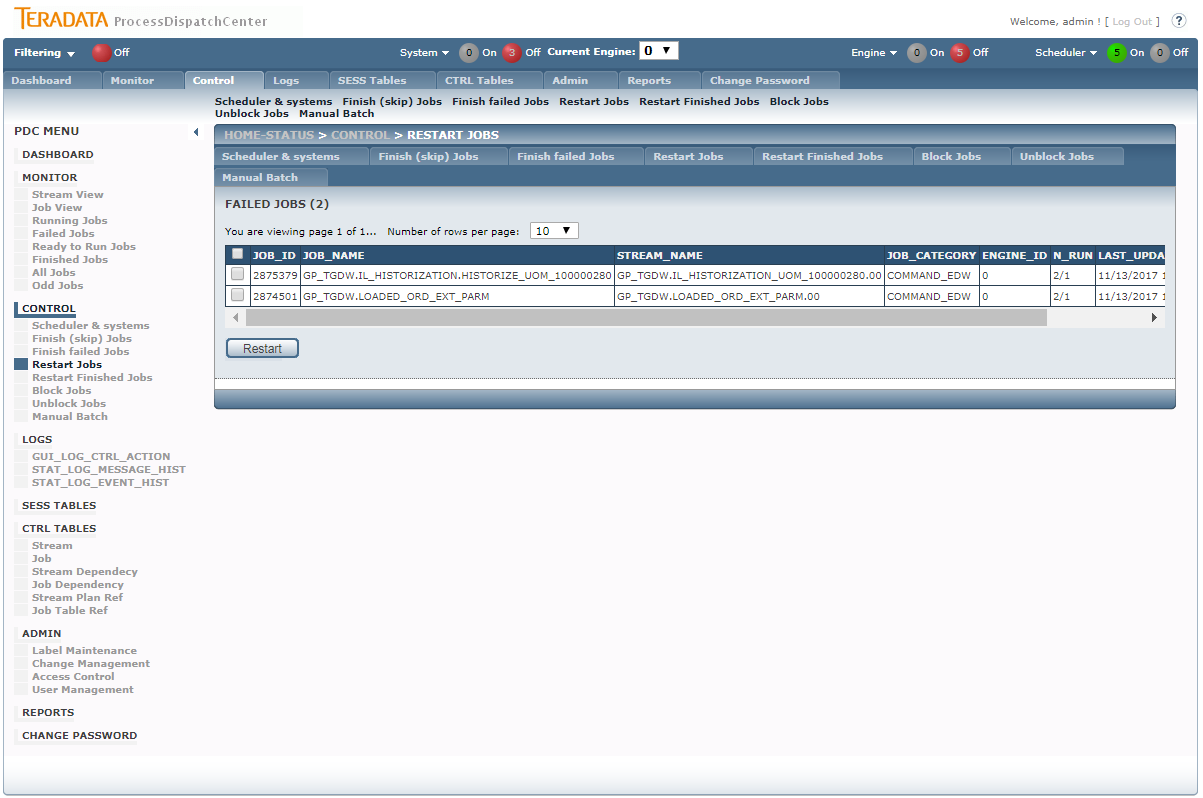


Figure 28 - GUI Control - Restart Jobs page

On the page you can select multiple or all jobs for processing. The action executed on jobs depends of value of always\_restart parameter and button which is pressed:

* Restart – job is returned to Scheduler by increasing value of max\_runs parameter by the value of max\_runs parameter from CTRL\_JOB table. The value of restart parameter is set to true even if resume of job is permited.
* Resume – job is returned to Scheduler by increasing value of max\_runs parameter by the value of max\_runs parameter from CTRL\_JOB table. The value of restart parameter is set to false only when value of always\_restart is set to false also. It means that resume has the same functionality as restart for jobs with value of always\_restart parameter set to true.

### Control – Restart Finished Jobs

This page is used for restarting already finished jobs.

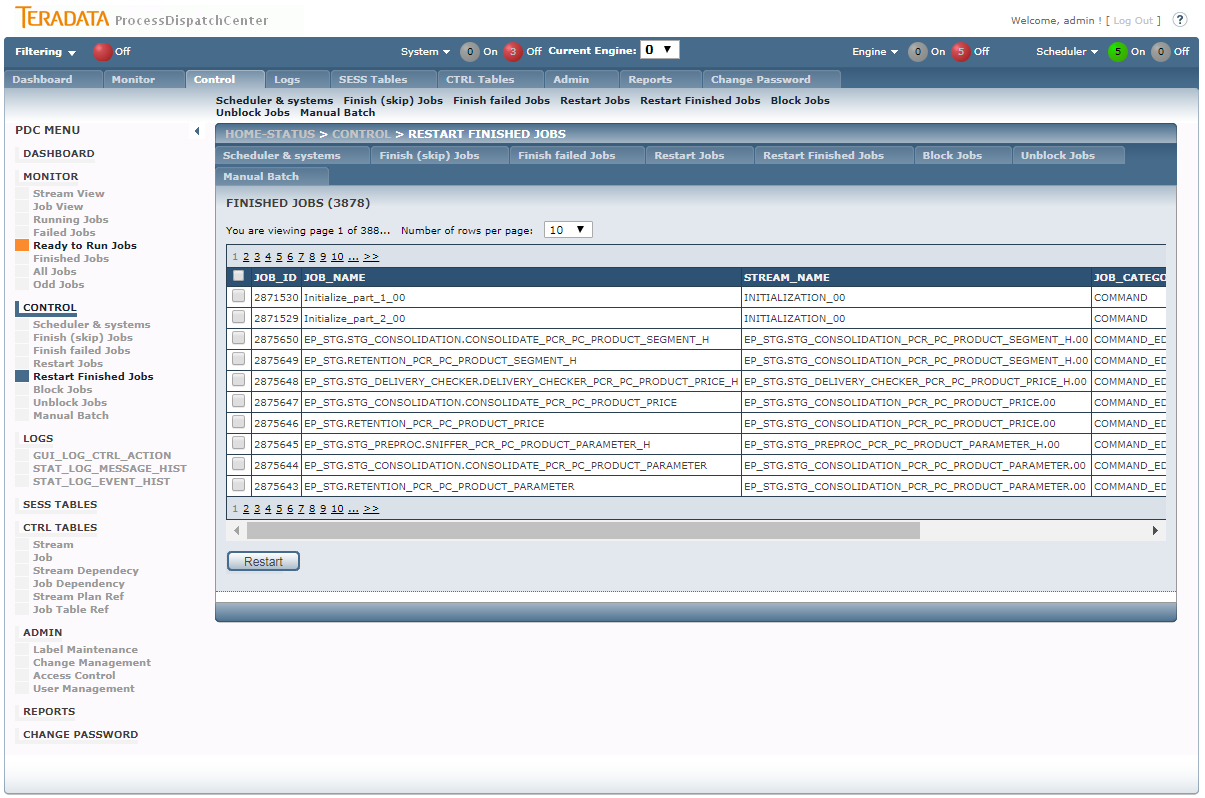


Figure 29 - GUI Control - Restart Finished Jobs page

Please note that even if multiple or all jobs can be selected, the jobs for restart has to be independent each other in case of job to job dependency therefore Scheduler doesn’t know anything about job’s dependency at this time.

### Control – Block Jobs

This page is used for blocking jobs.

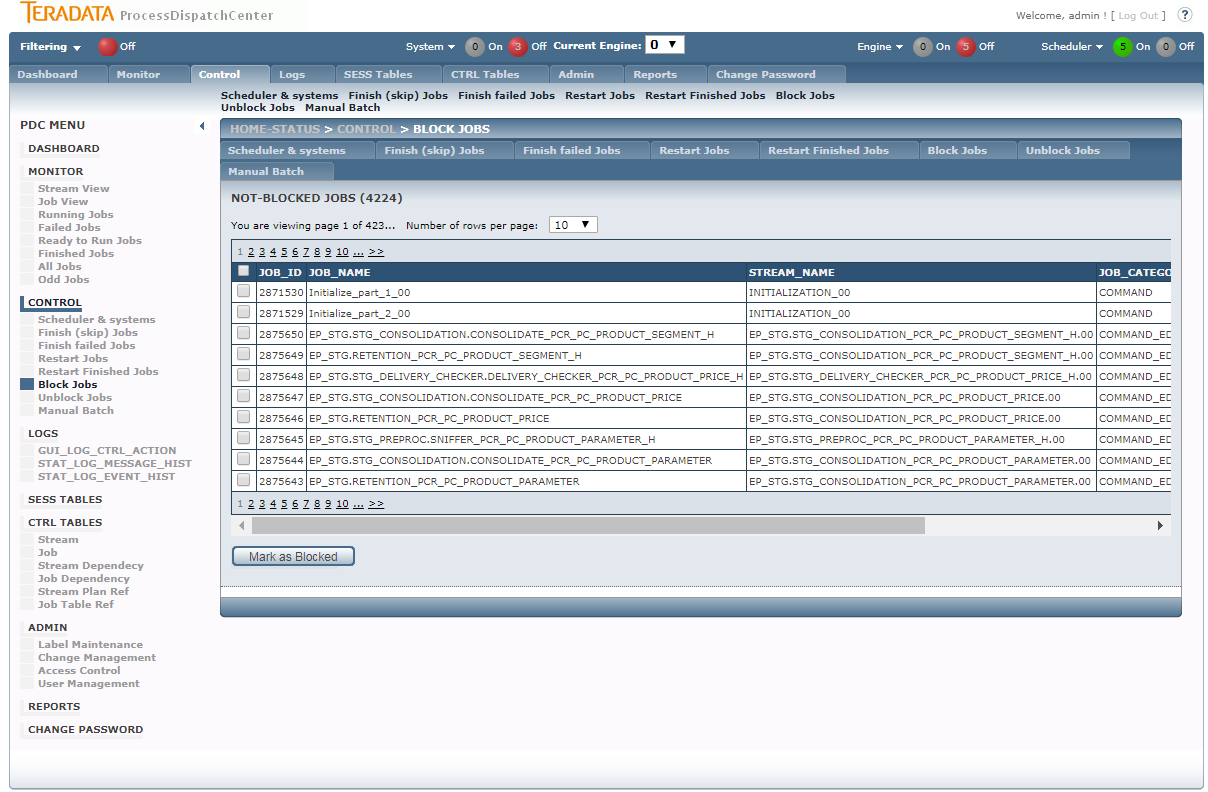


Figure 30 - GUI Control - Block Jobs page

Blocking functionality is used for temporarily removing some jobs from processing. It can be useful if error in processing is found and supervisor wants to protect wrong data to be processed by next jobs. In practice all not already blocked jobs can be blocked, but blocking has meaningful effect only on jobs not started yet. Therefore when finished or running job is selected for blocking, the not launched children has to be found for blocking instead of selected job.

### Control – Unblock Jobs

This page is used for unblocking jobs.

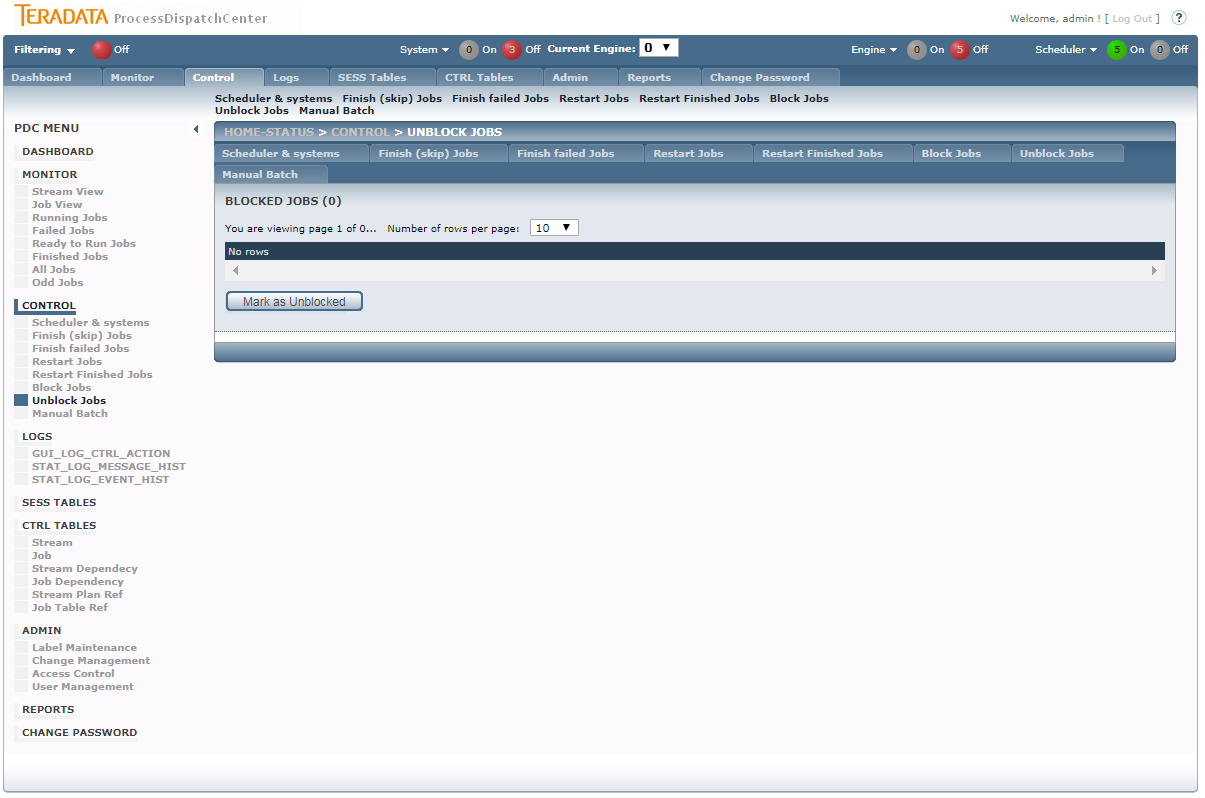


Figure 31 - GUI Control - Unblock Jobs page

Only blocked jobs are shown on page. Multiple or all jobs can be selected. Unblocking returns jobs its original status value.

### Control – Manual Batch

This page is used restarting group of jobs respecting its dependency. Load date for processing also can be chosen.

#### Step 1 – Select Date and Description

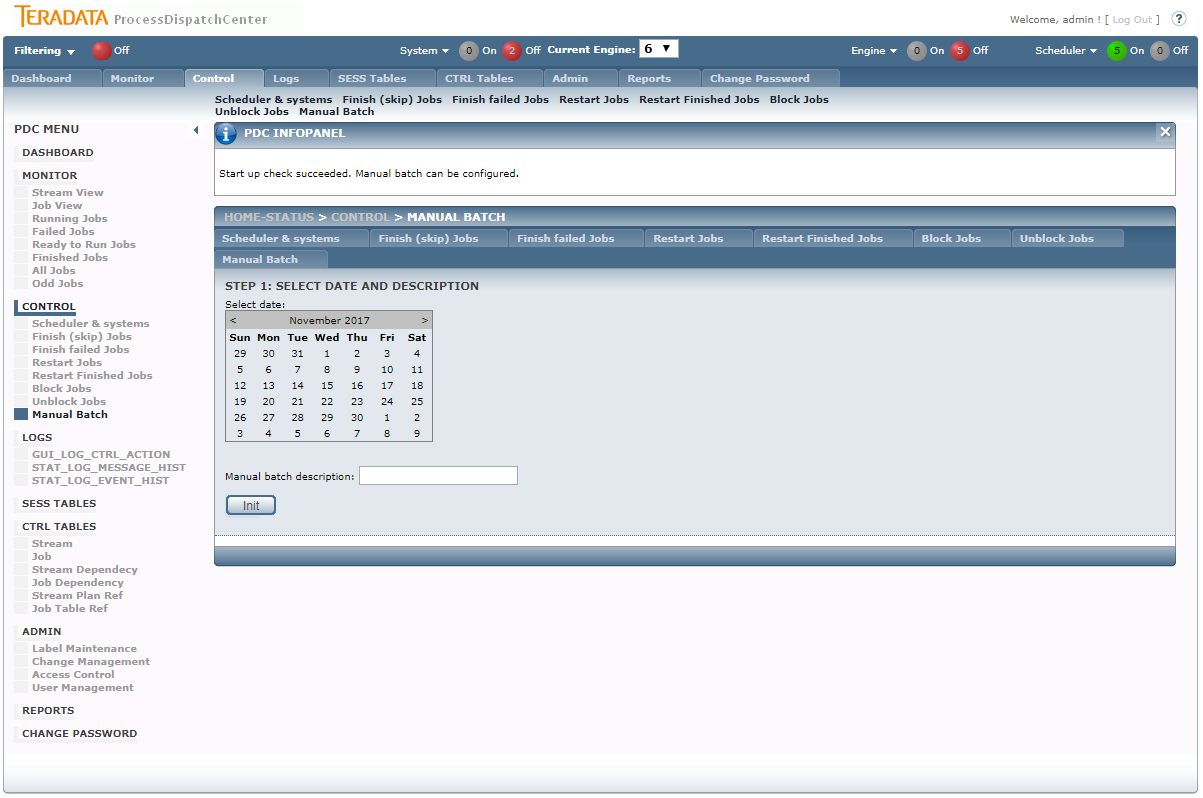


Figure 32 - GUI Control – Manual Batch – Select Date and Description page

PDC application has implemented several checks if initialization of Manual Batch is possible or not, therefore initialization of Manual Batch during standard or other Manual Batch processing can cause unexpected job’s processing. If Manual Batch initialization is allowed, the page for load date and description appears.

#### Step 2 – Select Jobs for Manual Batch

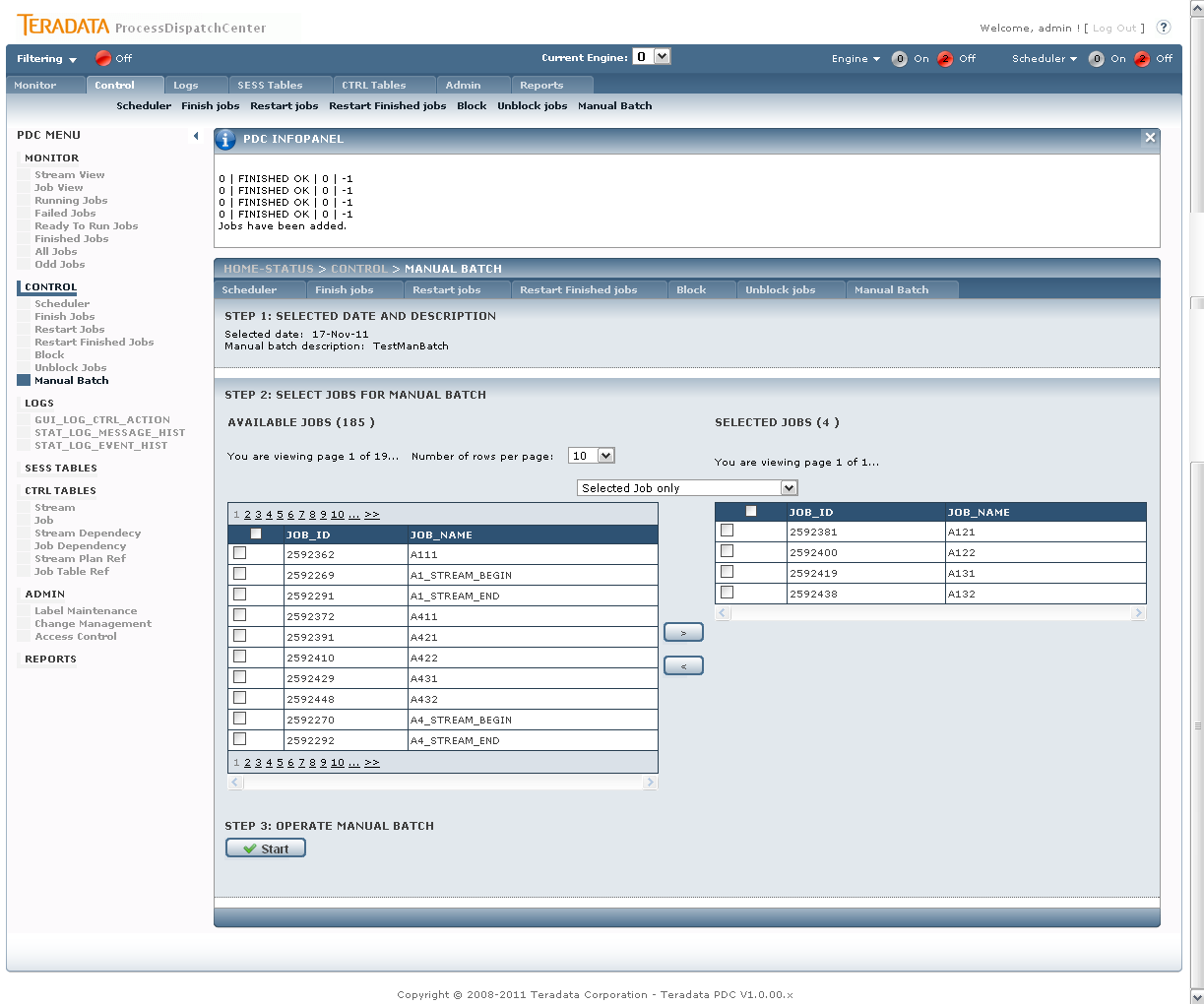


Figure 33 - GUI Control – Manual Batch – Select Jobs for Manual Batch page

On the page two tables are displayed. The left table contains jobs not selected for Manual Batch processing; the right one contains already selected jobs. There are four possibility how to get job from left table to the right one:

* Selected Job Only – the simplest possibility, you select job(s) in left table ane move them into right one. This option also works for reverse selection from right table to left one.
* Selected Job and Children – this option selects selected jobs and all its children. This option is useful when we have datafile and would like to select all jobs which process this datafile.
* Selected Job and Parents - this option selects selected job and all its parents. This option is useful when we want to refill some table and would like to select all jobs which prepared data for it.
* Selected Job and Child and Parents – union of the second and the third option.

#### Step 3 – Operate Manual Batch

There are two buttons available in this step. Pressing Start button causes Manual Batch processing contrary Cancel option finished Manual Batch without processing.

## Logs

Logs display controlling steps done on PDC application – it enables potential supervisor cooperation on problem solving, and also all problems captured by Framework checker application.

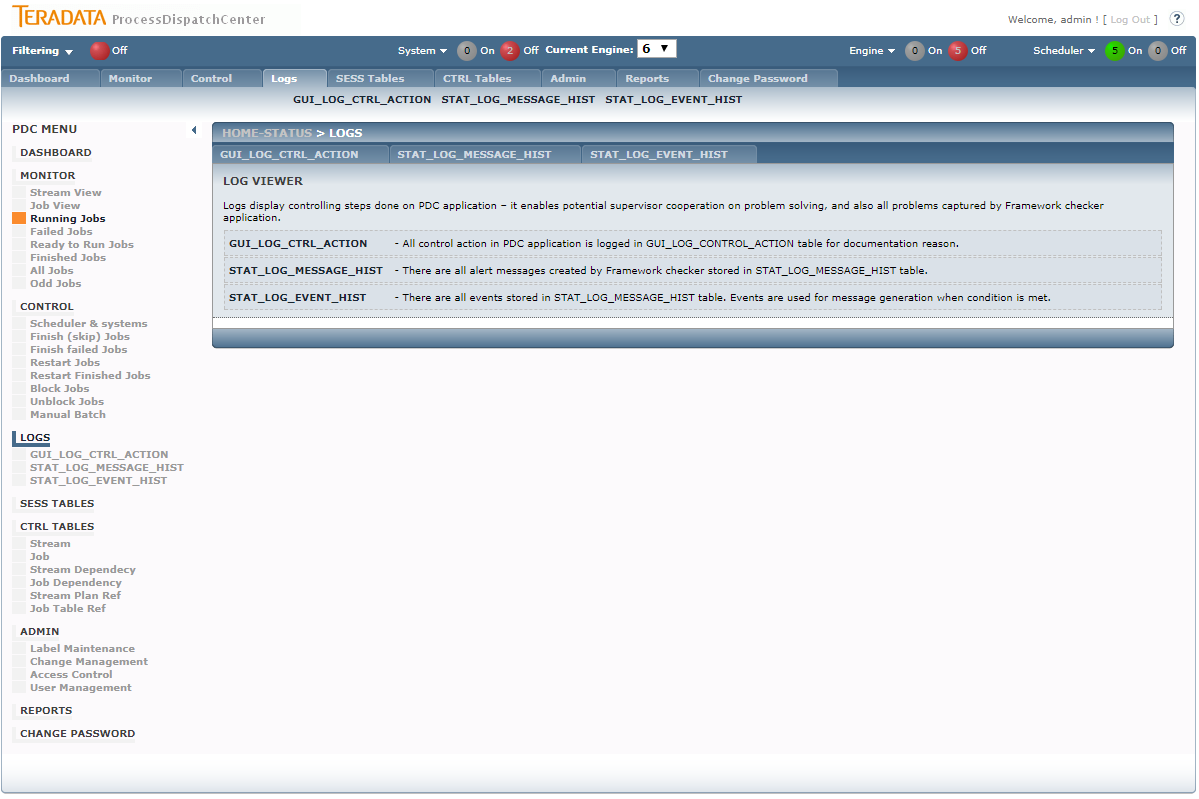


Figure 34 - GUI main Logs page

On Logs page functionality description is displayed only.

### Logs - GUI\_LOG\_CTRL\_ACTION

All control action in PDC application is logged in GUI\_LOG\_CONTROL\_ACTION table for documentation reason. Information as user name, action type, execution timestamp and SQL code are available.

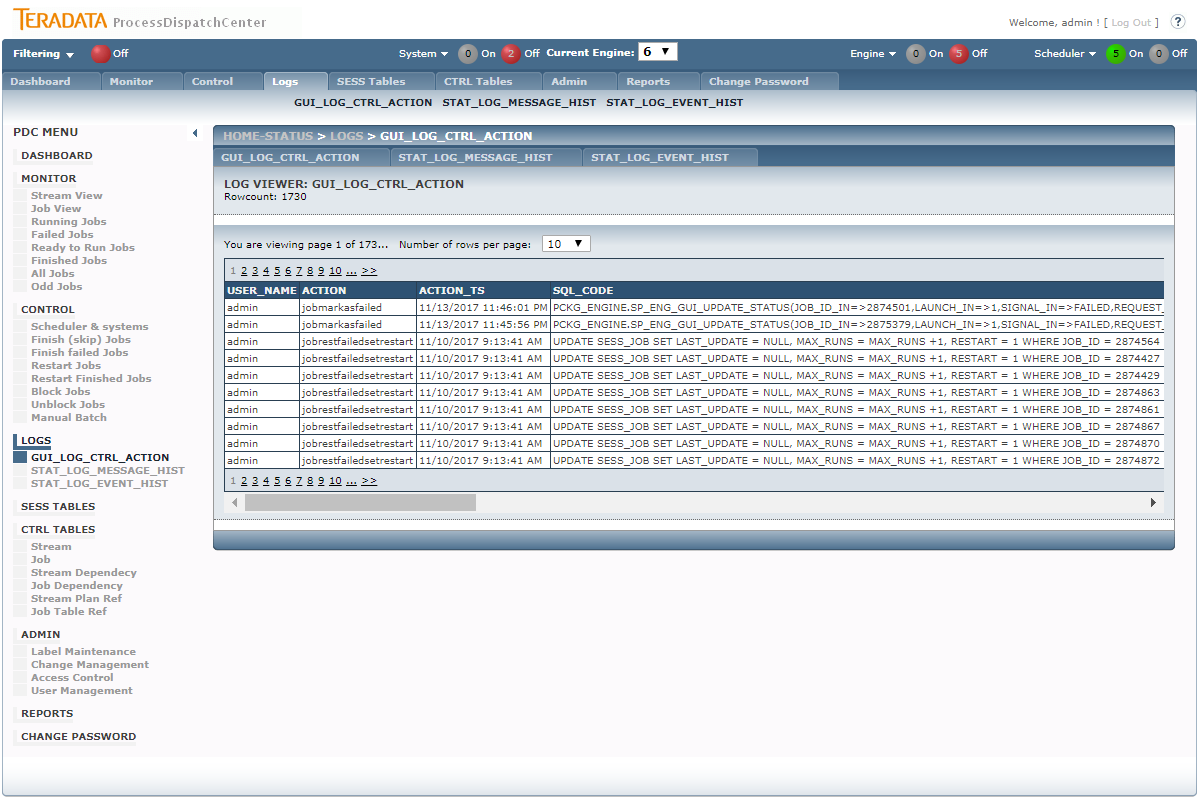


Figure 35 - GUI Logs – GUI\_LOG\_CTRL\_ACTION page

### Logs - STAT\_LOG\_MESSAGE\_HIST

There are all alert messages created by Framework checker stored in STAT\_LOG\_MESSAGE\_HIST table. The page allow simple checking table content.

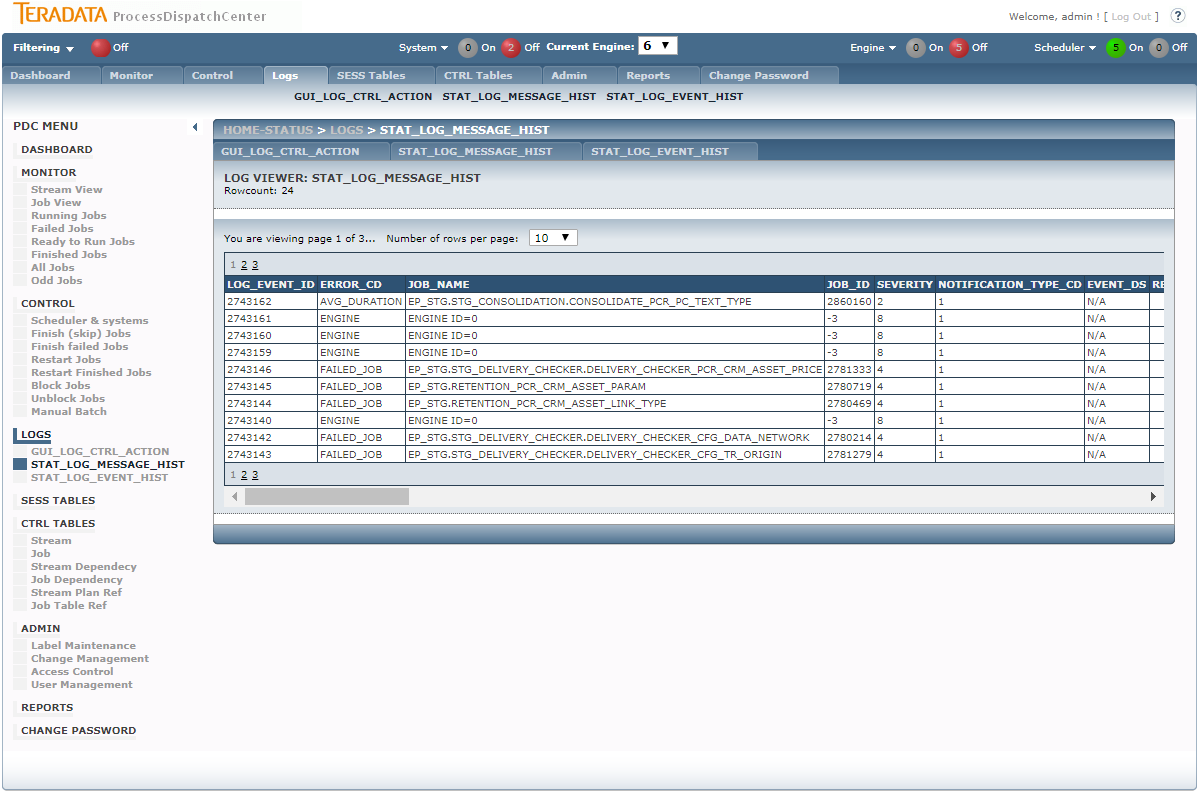


Figure 36 - GUI Logs - STAT\_LOG\_MESSAGE\_HIST page

### Logs - STAT\_LOG\_EVENT\_HIST

There are all events stored in STAT\_LOG\_MESSAGE\_HIST table. Events are used for message generation when condition is met. The page allow simple checking table content. Structure of Event page is similar as structure of Message page.

## SESS TABLES

SESS (session) tables contain operational metadata and store actual status of objects. SESS TABLES part enables changes in the table content. Such change can represent increasing actual job priority, command line changes and so on.

## CTRL TABLES

CTRL (control) tables store configuration metadata of jobs, streams, their dependences, locks and calendars. This part of GUI application is used for creating or changing this metadata. Application automatically checks correctness of typed values. Changes are enabled only when a label is selected which is necessary for change management process package creation. All changes are directly interpreted in table and also stored in GUI\_CHANGE\_CONTROL table for change management process. For editing only Edit and Delete option are available. There is general rule:

* When changing value of key column(s) of table, the new record creates
* When changing value of not key column(s) of table, only changed values are edited in existing record

Changing content of CTRL tables is meaningful only on development environment.

### CTRL TABLES – Stream

This page is used for making changes in CTRL\_STRAM table

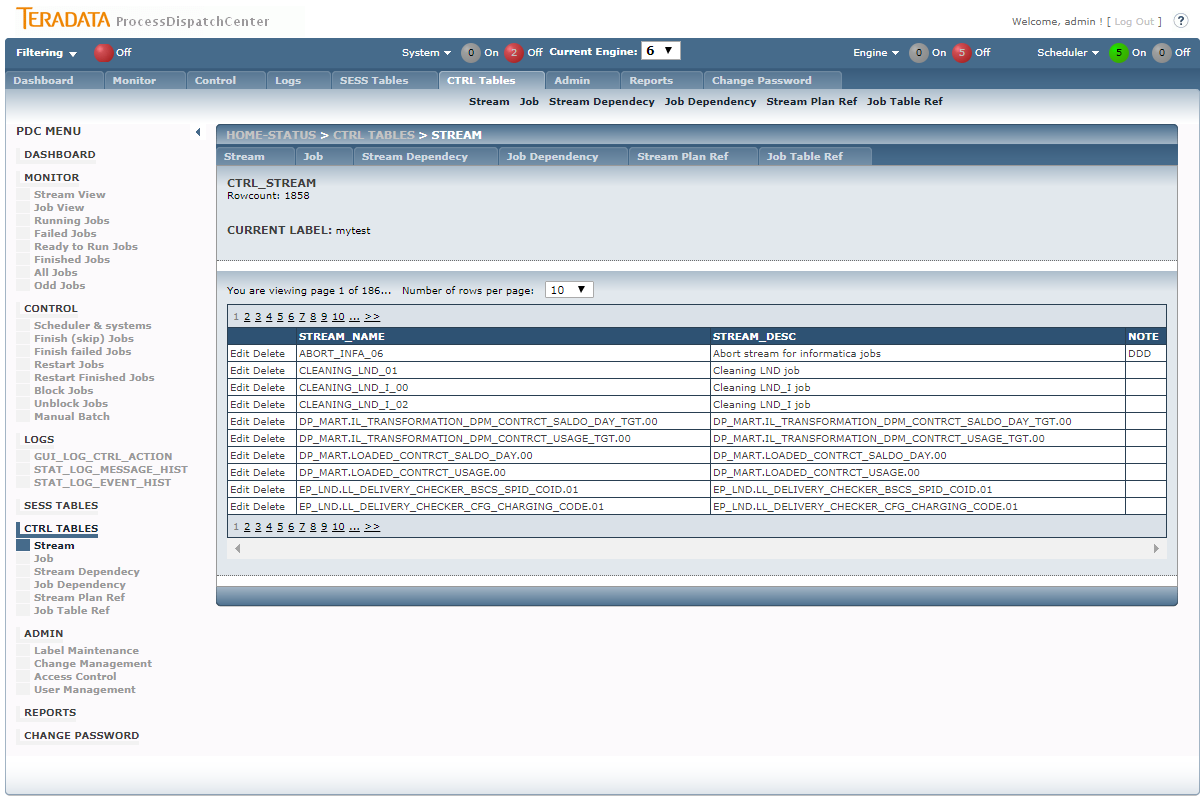


Figure 37 - GUI CTRL TABLES - Stream page

Columns and its meaning:

* **stream\_name** - Name of the stream, unique stream identifier
* **stream\_desc** - Description of the stream
* **note** - Note for stream procession

### CTRL TABLES – Job

This page is used for making changes in CTRL\_JOB table

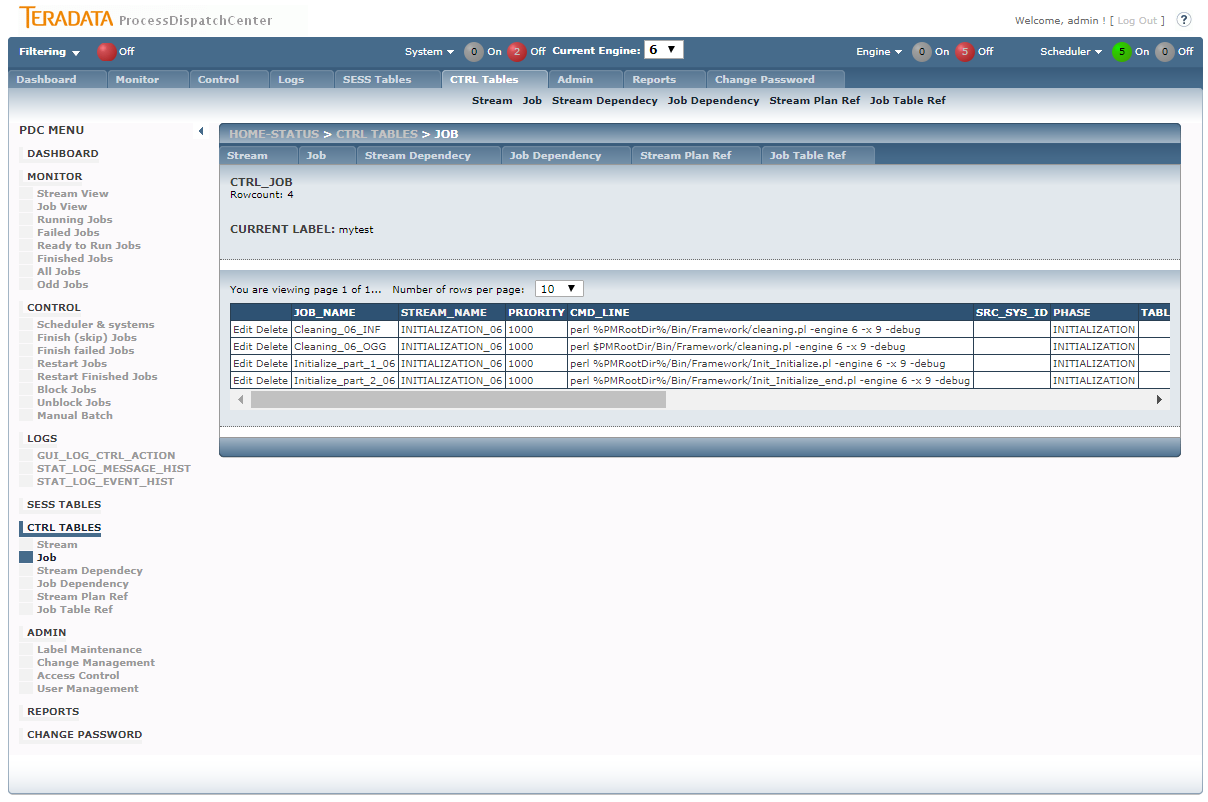


Figure 38 - GUI CTRL TABLES - Job page

Columns and its meaning:

* **job\_name** - Name of the job, unique job’s identifier.
* **stream\_name** - Name of the stream which job belongs to. Foreign key into CTRL\_STREAM table.
* **priority** - Priority of the job, less means more. Zero priority is used for system’s jobs which don’t respect paralell control.
* **cmd\_line** - Command line of the job.
* **src\_sys\_id** - Source system code which data is processed. Foreign key into CTRL\_SRC\_SYS table.
* **phase** - Phase name of DWH transformation. Values are lookuped from LKP\_PHASE table.
* **table\_name** - Main table name which data is processed.
* **job\_category** - Category of the job. Suitable for parallelism control. Values are lookuped from LKP\_JOB\_CATEGORY table.
* **job\_type** - Kind of job is used for special job handling. Values are lookuped from LKP\_JOB\_TYPE table.
* **cont\_anyway** – Set true if continue next job execution when job fails is asked.
* **max\_runs** - Maximum number of job launches without manual restart
* **always\_restart** – Set true when doing restart, even in case when resume is asked, is necessary
* **status\_begin** - Force pushed status when job launch, bypass calendar entry.
* **waiting\_hr** - Minimum number of hours passed from load\_date for launch granting
* **deadline\_hr** - Maximal number of hours passed from load\_date when job should be finished
* **engine\_id** - Engine id which can process the job
* **job\_desc** - Description for job
* **author** - Author of the job
* **note** - Note for job procession

### CTRL TABLES – Stream Dependency

This page is be used for making changes in CTRL\_STREAM\_DEPENDENCY table.

The records in CTRL\_STREAM\_DEPENDENCY define execution order of streams in processing. This table has child – parent structure. Also relation type can be defined, but standard stream’s dependency hasn’t specified this value.

Columns and its meaning:

* **stream\_name** - Child stream name. Foreign key into CTRL\_STREAM table.
* **parent\_stream\_name** - Parent stream name. Foreign key into CTRL\_STREAM table.
* **rel\_type** - Relation type, NULL for real stream dependency

### CTRL TABLES – Job Dependency

This page is be used for making changes in CTRL\_JOB\_DEPENDENCY table.

The records in CTRL\_JOB\_DEPENDENCY define execution order of jobs within the stream in processing. This table has child – parent structure. Also relation type can be defined, but standard job’s dependency hasn’t specified this value. Definition of job dependency within jobs in different stream is permitted but not recommended therefore it could not be done from GUI interface.

Columns and its meaning:

* **job\_name** - Child job name. Foreign key into CTRL\_JOB table.
* **parent\_job\_name** - Parent job name. Foreign key into CTRL\_JOB table.
* **rel\_type** - Relation type, NULL for real stream dependency

### CTRL TABLES – Stream Plan Ref

This page is be used for making changes in CTRL\_STREAM\_PLAN\_REF table.

The records in CTRL\_STREAM\_PLAN\_REF defines execution calendar for job’s launch. There is no limitation on number of calendar connected to one stream. System evaluates all calendar entry. For job’s launching is sufficient if only one calendar entry is active. The value of runplan is structured 9 characters long text lookuped from LKP\_CALENDAR table.

Columns and its meaning:

* **stream\_name** - Name of the stream
* **runplan** - Plan for job in stream launch [type][reg|irreg][3from][3to][shift]

Runplan definition ABCCCDDDE means:

* A – calendar type:
  + D – daily, used only fro irreguler entry
  + W – weekly – day in week Monday = 1, Sunday = 7
  + M – monthly – day in month
  + Y – yearly - day in year
* B – regular, iregular or counted calendar type
  + R – regular
  + I – iregular (for ultimo of month or year)
  + C – counted (e.g. the second Sunday in month)
* CCC – “from” value in R & I and count in C
* DDD – “to” value in R & I and date type in C
* E – execution shift:
  + A – after – if execution is not permited in load date (e.g. holiday), jobs will be executed somewhen in next days (not implemented, prepared for future use)
  + B – before – if execution is not permited in load date, jobs will be executed somewhen in prewious days (not implemented, prepared for future use)
  + E – exact – execution of jobs is done without recpecting holiday days
  + I – ignore – execution of jobs during holiday days is not done. Haliday days are counted all holiday and Saturdays and Sundays.

**Note:** definition of holiday is done via

**Examples:**

WR000000E – not exists -> jobs will be skipped

WR001007E – every day (from Monday (001) to Sunday (007))

WR001005E – every working day (from Monday (001) to Friday (005))

MR001001E – every first day in month (from (001) to (001))

MI000000E – every last day in month

WC002005E – every second (002) Friday (005) in month

### CTRL TABLES – Job Table Ref

This page is be used for making changes in CTRL\_JOB\_TABLE\_REF table.

Records in CTRL\_JOB\_TABLE\_REF define lock which job use on mentioned objects. Lock type should be “write” or “read”. There is possibility running many jobs having “read” on same object concurrently, but jobs having “write” lock has to run separate. If job uses “write” and “read” lock on the same object, only “write” log is entered.

Columns and its meaning:

* **job\_name** - Name of the job, foreign key into CTRL\_JOB table
* **database\_name** - Database or directory name which job in which object is located.
* **table\_name** - Table or file name which job is working with. Sign “%” can be used for selecting object using “LIKE” functionality.
* **lock\_type** - Lock type, W=write, R=read

## Admin

Admin section contains necessary administrative tasks such as label maintenance, change management process, access control and so on.

### Admin – Label maintenance

Labels are used for identification of metadata changes which is necessary for correctness of change management process.

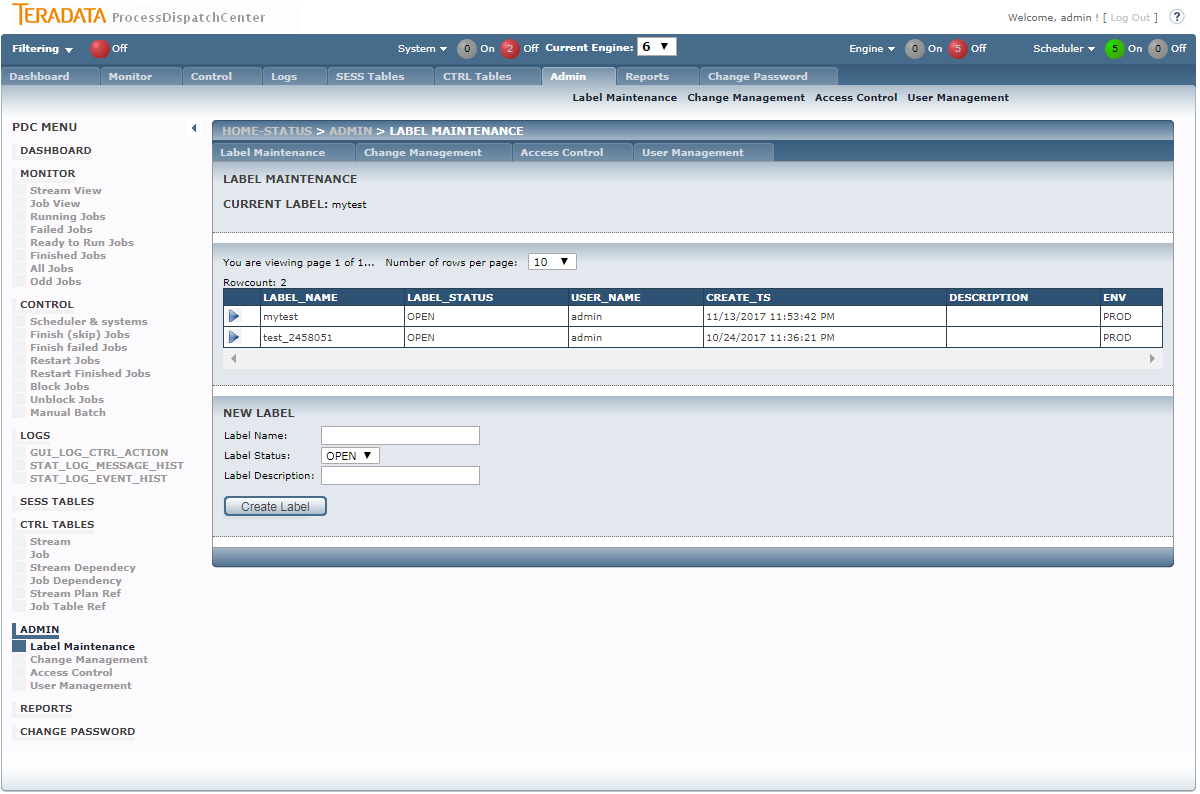


Figure 39 - GUI Admin – Label maintenance page

In middle section of the page the label selection is permitted. Only opened labels can be chosen. Bottom section can be used for new label creation. Only label name and description can be entered. If some label is already chosen, the label name is displayed in top section.

### Admin – Change Management

Change management creates SQL file with all changes made in CTRL tables labeled with chosen label. This file is used for tracking these changes in test and production environment.

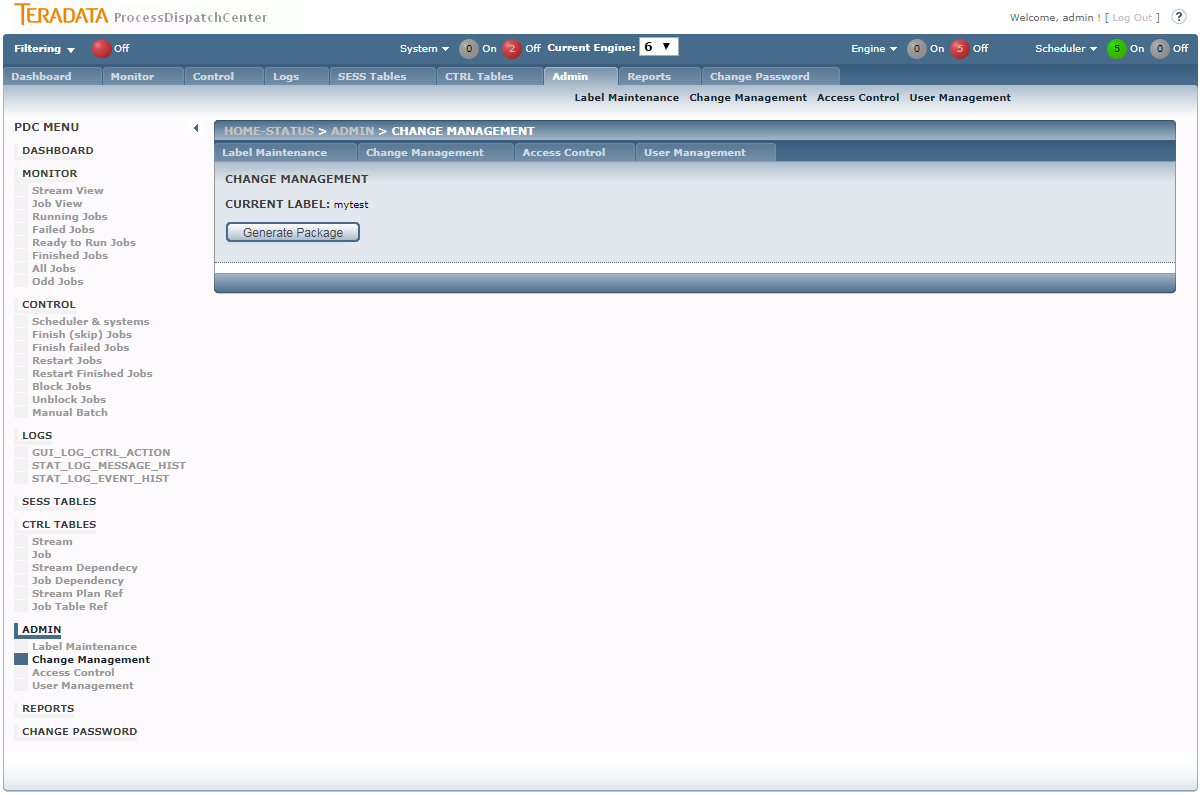


Figure 40 - GUI Admin – Change Management page

When pushing Generate Package button, process will ask you for file name for script save.

### Admin – Access Control

Access Control page is used for access rights granting. Generally there are two types of rights related to page:

* SHOW – user can see page content
* CONTROL – function buttons are active

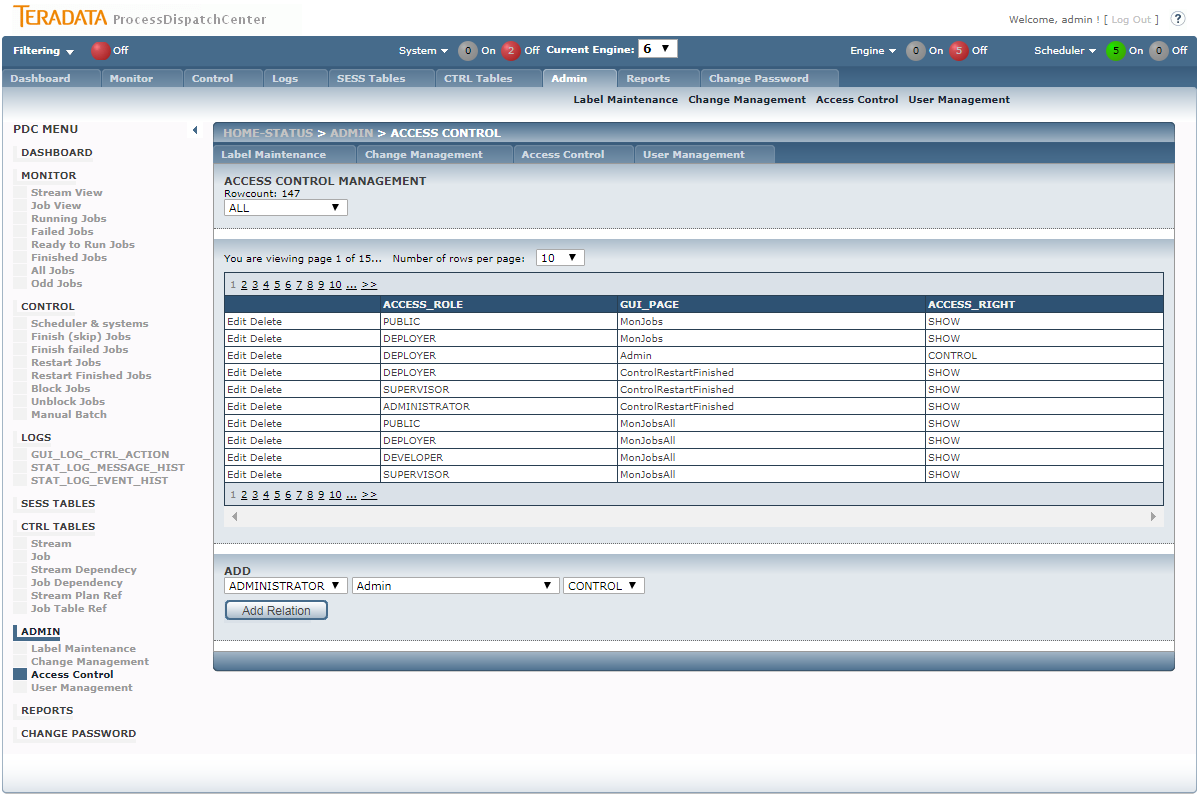


Figure 41 - GUI Admin – Access Control page

The rights are not granted on user level, but on group level. Every user with access to PDC application has to be a part of some group.

## Reports

Reports page is a signpost for customer reports related for PDC processing. Customers can simply use their reporting tools for creating reports based on PDC metadata and place them on this page.

# Metadata repository tables

All PDC application based metadata are stored in tables at Oracle database. Main table’s structure and vales meaning is described in this section.

Tables of PDC application can be sorted in next classes:

* CTRL tables – controlling metadata
* STAT tables – historical storage of operational metadata
* SESS tables – operational metadata, current state
* LKP tables – lookups
* GUI tables – tables for GUI application support
* CUST tables – customer specific tables

Most important tables from user perspective are listed below:

## CTRL tables

### CTRL\_STREAM table

**Stream definition**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| STREAM\_NAME | VARCHAR2(128) | N | Name of the stream |
| STREAM\_DESC | VARCHAR2(2048) | Y | Description of the stream |
| NOTE | VARCHAR2(2048) | Y | Note for stream procession |

### CTRL\_STREAM\_DEPENDENCY table

**Dependency between streams**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| STREAM\_NAME | VARCHAR2(128) | N | Child stream name |
| PARENT\_STREAM\_NAME | VARCHAR2(128) | N | Parent stream name |
| REL\_TYPE | CHAR(1) | Y | Relation type, NULL for real stream dependency |

### CTRL\_STREAM\_PLAN\_REF table

**Calendars to stream assignation**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| STREAM\_NAME | VARCHAR2(128) | N | Name of the stream |
| RUNPLAN | CHAR(9) | N | Plan for job in stream launch [type][reg|irreg][3from][3to][shift] |
| COUNTRY\_CD | VARCHAR2(4) | Y | Country code |

### CTRL\_JOB table

**Job definition**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| JOB\_NAME | VARCHAR2(128) | N | Name of the job |
| STREAM\_NAME | VARCHAR2(128) | N | Name of the stream which job belongs to |
| PRIORITY | NUMBER(38,0) | Y | Priority of the job, less number means higher priority |
| CMD\_LINE | VARCHAR2(1024) | Y | Command line of the job |
| SRC\_SYS\_ID | NUMBER(38,0) | Y | Source system code which data is processed |
| PHASE | VARCHAR2(32) | Y | Phase name of DWH transformation |
| TABLE\_NAME | VARCHAR2(128) | Y | Main table name which data is processed |
| JOB\_CATEGORY | VARCHAR2(32) | N | Category of the job. Used for parallelism control |
| JOB\_TYPE | VARCHAR2(32) | Y | Kind of job |
| TOUGHNESS | VARCHAR2(32) | Y | Job toughness |
| CONT\_ANYWAY | NUMBER(38,0) | N | If continue next job execution when job fails |
| MAX\_RUNS | NUMBER(38,0) | N | Maximum number of job launches without manual restart |
| ALWAYS\_RESTART | NUMBER(38,0) | N | Do restart in any case |
| STATUS\_BEGIN | NUMBER(38,0) | Y | Force pushed status when job launch |
| WAITING\_HR | NUMBER(38,0) | Y | Minimum number of hours passed from load\_date for launch granting |
| DEADLINE\_HR | NUMBER(38,0) | Y | Maximal number of hours passed from load\_date when job should be finished |
| ENGINE\_ID | NUMBER(38,0) | N | Identification of engine which can process the job |
| JOB\_DESC | VARCHAR2(1024) | Y | Job description |
| AUTHOR | VARCHAR2(64) | Y | Author of the job |
| NOTE | VARCHAR2(2048) | Y | Note for job procession |

More details about columns:

* **job\_name** – name of job, unique job identification
* **stream\_name** – job to stream relation, foreign key into CTRL\_STREAMtable
* **priority** – job’s priority, lesser is higher. Priority lesser then 1000 is reserved for dynamical priority assign (future use). Zero priority using some service jobs which can’t respect parallel control
* **cmd\_line** – command line
* **src\_sys\_id**  - source system identification in CTRL\_SRC\_SYS table
* **phase** – lookuped from LKP\_PHASE table
* **table\_name** – main table name which data is transferred by job
* **job\_category** – job category is used for parallelism control (task\_typein CTRL\_TASK\_PARAMETERS table). Value is lookuped form LKP\_JOB\_CATEGORY table
* **job\_type** – job type is used for special job behavior handling, lookuped from LKP\_JOB\_TYPE table.
* **cont\_anyway** – this value is used for decision if finish job when fails after all possible automatics restart (defined by MAX\_RUNS value). If parameter is set, the failed finished jobs are identified by VOID\_FINISHED status value. “0” value is used for standard jobs, “1” is used only for jobs which success execution is not necessary (compress log files and so on).
* **max\_runs** – maximal number of automatically drive job launching. If all attempt are already consumed and job is not finished, manual job handling is required. There is a delay in minutes defined in CTRL\_JOB\_STATUS table between each attempt of job restart.
* **always\_restart** – value defines restart type when job failed. When “1” is set, both restart and resume action causes restart. Restart type has to be supported by application called from job.
* **status\_begin** – suggested status for job. If value is not NULL, the calendar entry is ignored.
* **waiting\_hr** – the value of number of hours measured from “load\_date” value which has pass before job can be launched.
* **deadline\_hr** – the value of number of hours measured from “load\_date” value when is asked ending of job (job SLA definition)
* **engine\_id** – defines which Engine job belongs, default value is 0 (standard processing).
* **job\_desc** – job’s description
* **author** – job’s creator
* **note** – note for job’s execution

### CTRL\_JOB\_DEPENDENCY table

**Dependency between jobs inside the stream (dependency between jobs in different stream is permitted but not recomended.)**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| JOB\_NAME | VARCHAR2(128) | N | Child job name |
| PARENT\_JOB\_NAME | VARCHAR2(128) | N | Parent job name |
| REL\_TYPE | CHAR(1) | Y | Relation type, NULL for real job dependency |

### CTRL\_JOB\_TABLE\_REF table

**Relation of which job is using which table**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| JOB\_NAME | VARCHAR2(128) | N | Name of the job |
| DATABASE\_NAME | VARCHAR2(128) | N | Database name which job is working with |
| TABLE\_NAME | VARCHAR2(128) | N | Table name which job is working with |
| LOCK\_TYPE | CHAR(1) | N | Lock type, W=write, R=read |

## SESS tables

### SESS\_JOB (SESS\_JOB\_BCKP) table

**List of the job with actual status of processing**

**Table is filled with initial status of jobs during initialization process.** initial status before load start is backed up in SESS\_JOB\_BCKP table**.**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| JOB\_ID | NUMBER(38,0) | N | Job identification |
| STREAM\_ID | NUMBER(38,0) | N | Unique sequence number generated for stream\_name identification |
| JOB\_NAME | VARCHAR2(128) | N | Name of the job |
| STREAM\_NAME | VARCHAR2(128) | N | Name of the stream which job belongs to |
| STATUS | NUMBER(38,0) | N | Current status of the job |
| LAST\_UPDATE | TIMESTAMP(6)(11,6) | Y | Timestamp of the job status last update |
| LOAD\_DATE | DATE(7) | N | Load date |
| PRIORITY | NUMBER(38,0) | Y | Priority of the job, less number means higher priority |
| CMD\_LINE | VARCHAR2(1024) | Y | Command line of the job |
| SRC\_SYS\_ID | NUMBER(38,0) | Y | Source system code which data is processed |
| PHASE | VARCHAR2(32) | Y | Phase name of DWH transformation |
| TABLE\_NAME | VARCHAR2(128) | Y | Main table name which data is processed |
| JOB\_CATEGORY | VARCHAR2(32) | N | Category of the job. Suitable for parallelism control |
| JOB\_TYPE | VARCHAR2(32) | Y | Kind of job |
| TOUGHNESS | NUMBER(38,0) | Y | Job toughness |
| CONT\_ANYWAY | NUMBER(38,0) | N | If continue next job execution when job fails |
| RESTART | NUMBER(38,0) | N | Do a restart |
| ALWAYS\_RESTART | NUMBER(38,0) | N | Do restart in any case |
| N\_RUN | NUMBER(38,0) | N | Actual number of job launch |
| MAX\_RUNS | NUMBER(38,0) | N | Maximum number of job launches without manual restart |
| WAITING\_HR | NUMBER(38,0) | Y | Minimum number of hours passed from load\_date for launch granting |
| DEADLINE\_HR | NUMBER(38,0) | Y | Maximal number of hours passed from load\_date when job should be finished |
| APPLICATION\_ID | NUMBER(38,0) | Y | Application identification which drive the job |
| ENGINE\_ID | NUMBER(38,0) | N | Engine id which can process the job |
| SYSTEM\_NAME | VARCHAR2(128) | Y | System which process the job |
| QUEUE\_NUMBER | NUMBER(38,0) | Y | Queue number |

The column meaning is same as in CTRL\_JOB, but there are also some additional columns:

* **job\_id** – job’s sequence number, unique job identifier in table
* **stream\_id** – stream’s sequence number
* **status** – job’s actual status. Value and meaning are taken from CTRL\_JOB\_STATUS table.
* **last\_update** – timestamp of the last status change, NULL value means year 2000
* **load\_date** – business date which data is transferred by job
* **restart** – set to “1” if restart of job is required instead of resume
* **n\_run** – actual number of job launching attempts
* **application\_id** – application identification which changed job’s status
* **system\_name –** system / ETL server which run the job
* **queue\_number** – queue unique identificationwhich is actually processing job

### SESS\_JOB\_DEPENDENCY (SESS\_JOB\_DEPENDENCY\_BCKP) table

**Current state of job dependencies**

During initialization process initial state of dependencies is calculated from CTRL\_STREAM\_DEPENDENCY and CTRL\_JOB\_DEPENDENCY table. Initial state is backed up in SESS\_JOB\_DEPENDENCY\_BCKP table.

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| JOB\_ID | NUMBER(38,0) | N | Child job name identification |
| JOB\_NAME | VARCHAR2(128) | N | Name of the child job |
| PARENT\_JOB\_ID | NUMBER(38,0) | N | Parent job name identification |
| PARENT\_JOB\_NAME | VARCHAR2(128) | N | Name of the parent job |
| REL\_TYPE | CHAR(1) | Y | Relation type, NULL for real job dependency |
| ENGINE\_ID | NUMBER(22) | Y | Engine identification |

### SESS\_QUEUE table

**Queue list for job execution**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| QUEUE\_NUMBER | NUMBER(38,0) | N | Queue number |
| JOB\_NAME | VARCHAR2(128) | Y | Job name running in queue number |
| JOB\_ID | NUMBER(38,0) | Y | Job identification running in queue number |
| AVAILABLE | NUMBER(38,0) | N | Is queue number available for job launch? |
| LAST\_UPDATE | TIMESTAMP(6)(11,6) | Y | Timestamp of last used |
| ENGINE\_ID | NUMBER(38,0) | N | Engine id for which is queue determine |
| SYSTEM\_NAME | VARCHAR2(128) | Y | Name of the system / ETL server on which is job running |
| RUNNING\_JOB\_PID | NUMBER(22,0) | Y | PID of PDC run job process |

### SESS\_STATUS table

**Log table for job status trace**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| JOB\_ID | NUMBER(38,0) | N | Job identification |
| JOB\_NAME | VARCHAR2(128) | N | Name of the job |
| STREAM\_NAME | VARCHAR2(128) | N | Name of the stream |
| STATUS\_TS | TIMESTAMP(6)(11,6) | N | Timestamp of status change |
| LOAD\_DATE | DATE(7) | N | Load date |
| STATUS | NUMBER(38,0) | N | Job status |
| N\_RUN | NUMBER(38,0) | N | Number of job launching |
| SIGNAL | VARCHAR2(16) | N | Signal which changed the status |
| APPLICATION\_ID | NUMBER(38,0) | Y | Application identification which drives the job |
| ENGINE\_ID | NUMBER(38,0) | N | Engine ID |
| SYSTEM\_NAME | VARCHAR2(128) | Y | System name where status change was applied |

### SESS\_JOB\_TABLE\_REF table

**Relation of which job is using which table**

Fill during initialization process with rows valid for executed jobs in load.

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| JOB\_NAME | VARCHAR2(128) | N | Name of the job |
| DATABASE\_NAME | VARCHAR2(128) | N | Database name which job is working with |
| TABLE\_NAME | VARCHAR2(128) | N | Table name |
| LOCK\_TYPE | CHAR(1) | N | Lock type, W=write, R=read |
| JOB\_NAME | VARCHAR2(128) | N | Name of the job |
| DATABASE\_NAME | VARCHAR2(128) | N | Database name which job is working with |
| TABLE\_NAME | VARCHAR2(128) | N | Table name |
| LOCK\_TYPE | CHAR(1) | N | Lock type, W=write, R=read |
| JOB\_NAME | VARCHAR2(128) | N | Name of the job |
| DATABASE\_NAME | VARCHAR2(128) | N | Database name which job is working with |
| TABLE\_NAME | VARCHAR2(128) | N | Table name |

### SESS\_PARALLELISM\_CONTROL

**Current state of parallelism control**

Refreshed by engine via SP\_ENG\_GET\_JOB\_LIST procedure

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| PARAM\_VAL\_INT\_CURR | NUMBER(38,0) | Y | toughness sum of currently running jobs |
| PARAM\_VAL\_INT\_MAX | NUMBER(38,0) | Y | Maximal threshold which can be reached by category or subcategory |
| PARAM\_VAL\_INT\_DEFAULT | NUMBER(38,0) | N | Default maximal threshold for category or subcategory |
| JOB\_CATEGORY | VARCHAR2(128) | Y | Category of the job. Suitable for parallelism control |
| PARENT\_JOB\_CATEGORY | VARCHAR2(128) | Y | Parent job category for parallelism control |
| ENGINE\_ID | NUMBER(38,0) | N | Engine identification |
| SYSTEM\_NAME | VARCHAR2(128) | Y | System which process the job |
| OVERLOAD | NUMBER(22,0) | Y | If overload process was done |
| CURR\_RUNNING\_JOBS | NUMBER(22,0) | Y | Current running job counter |
| TOTAL\_TOUGH | NUMBER(22,0) | Y | Total toughness on category |
| TASK\_MIN\_CONTROL | NUMBER(22,0) | Y | Min. num of jobs when toughness reservation is applied |

## PDC config tables

### CTRL\_JOB\_STATUS table

**Lookup of job statuses**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| STATUS | NUMBER(38,0) | N | Status of the job |
| RUNABLE | VARCHAR2(16) | N | Status category |
| DELAY\_MINUTES | NUMBER(38,0) | N | Number in minutes how long job has to stay in status before launching |
| FINISHED | NUMBER(38,0) | N | Is the status final ? |
| FINISHED\_SUCCESSFULLY | NUMBER(38,0) | N | Was the job in this status finished successfully? |
| EXECUTABLE | NUMBER(38,0) | N | Is status really executable |
| SORTING\_ORDER | NUMBER(38,0) | N | Sorting order (for stream status indication) |
| DESCRIPTION | VARCHAR2(1024) | Y | Description of the status |

### CTRL\_NBD\_CAL table

**National bank holiday definition**

It’s used for calendar evaluation.

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| COUNTRY\_CD | VARCHAR2(4) | N | Country code |
| NBD\_DATE | DATE(7) | N | National bank holiday date |

### CTRL\_NEXT\_STATUS table

**Job status transition table**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| STATUS\_IN | NUMBER(38,0) | N | Input status |
| STATUS\_OUT | NUMBER(38,0) | N | Output status |
| SIGNAL | VARCHAR2(16) | N | Signal which changing the status |
| REQUEST | VARCHAR2(16) | N | Request for status change |
| LAUNCH | NUMBER(38,0) | N | Job is launching |
| CONT\_ANYWAY | NUMBER(38,0) | N | If continue next job execution when job fails |

### CTRL\_TASK\_PAREMETERS table

**Parallelism control settings table**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| PARAM\_NAME | VARCHAR2(128) | Y | Parameter name |
| PARAM\_TYPE | VARCHAR2(128) | Y | Parameter type |
| PARAM\_VAL\_INT\_CURR | NUMBER(38,0) | Y | Not used, moved to SESS\_PARALLELISM\_CONTROL |
| PARAM\_VAL\_INT\_MAX | NUMBER(38,0) | Y | Maximal threshold which can be reached by category or subcategory |
| PARAM\_VAL\_INT\_DEFAULT | NUMBER(38,0) | N | Default maximal threshold for category or subcategory |
| PARAM\_DIMENSION | VARCHAR2(128) | Y | Used for TOUGH\_SPEC\_CATEGORY\_CONTROL - definition of phase |
| TASK\_TYPE | VARCHAR2(128) | Y | Job category or subcategory for parallelism control |
| PARENT\_TASK\_TYPE | VARCHAR2(128) | Y | Job category for parallelism control |
| VALID\_FROM | NUMBER(38,0) | Y | From which hour is values valid |
| VALID\_TO | NUMBER(38,0) | Y | To which hour is values valid |
| ENGINE\_ID | NUMBER(38,0) | Y | Engine identification |
| SYSTEM\_NAME | VARCHAR2(128) | Y | Name of the system / ETL server for which the parameter is valid |
| DESCRIPTION | VARCHAR2(1024) | Y | Description of parameter |

### CTRL\_PARAMETERS table

**Parameters settings**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| PARAM\_NAME | VARCHAR2(128) | N | Parameter name |
| PARAM\_CD | NUMBER(38,0) | Y | Parameter sequence code, mostly used for engine\_id |
| PARAM\_TYPE | VARCHAR2(128) | Y | Parameter type |
| PARAM\_VAL\_INT | NUMBER(38,0) | Y | Parameter value - number domain |
| PARAM\_VAL\_CHAR | VARCHAR2(1024) | Y | Parameter value - char domain |
| PARAM\_VAL\_DATE | DATE(7) | Y | Parameter value - date domain |
| PARAM\_VAL\_TS | TIMESTAMP(8) WITH LOCAL TIME ZONE(11,8) | Y | Parameter value - timestamp domain |
| DESCRIPTION | VARCHAR2(1024) | Y | Description of parameter meaning |

#### Parameters

| COLUMN\_NAME | Param  type | Param val | Engine  specific | System  specific | Mandatory | Default  value | Description |
| --- | --- | --- | --- | --- | --- | --- | --- |
| APPLICATION\_ID | SCHEDULER | INT | Y | N | Y | 0 | Actual value of application\_id used for standard processing (lookup table - LKP\_APPLICATION). |
| AVG\_DURATION\_FACTOR\_PERCENT | CHECKER | INT | N | N | Y | 150 | Maximal acceptable percentage of job run duration before alert is raised (100 = calculated expected duration). |
| AVG\_DURATION\_TOLERANCE | CHECKER | INT | N | N | Y | 1800 | Maximal acceptable tolerance of job run duration in seconds before alert is raised. Used for job notification settings when new job is created. |
| AVG\_END\_TM\_TOLERANCE | CHECKER | INT | N | N | Y | 3600 | Absolute acceptable tolerance of job end time in seconds before alert is raised. Used for job notification settings when new job is created. |
| DISPLAY\_ENG\_IN\_DASHBOARD | GUI | INT | N | N | N | 1 | Distinguisher if engine is shown in PDC dashboard. By default is shown. |
| ENGINE\_CATEGORY | GUI | CHAR | Y | N | N |  | Engine category used for categorizing of engine in Dashboard |
| ENGINE\_CONTROL | SCHEDULER | INT | Y | N | Y | 0 | If (1) system in param\_val\_char is active. Other systems are waiting. If (0) there is no system active. |
| ENGINE\_NAME | GUI | INT | Y | N | Y |  | Engine name shown in GUI. |
| ENGINE\_PROBLEM\_COUNT | CHECKER | INT | N | N | N | 1 | Engine issue event counter threshold. Alert is raised only If counter value is higher than threshold. |
| ENGINE\_STATUS | SCHEDULER | Timestamp | Y | Y | Y |  | Last activity timestamp of system. It's refreshed after every get job list cycle. |
| ENVIRONMENT | SCHEDULER | CHAR | N | N | Y |  | Environment shown in GUI header. |
| GUI\_COLOUR | GUI | CHAR | N | N | N |  | Color used for grid headers in GUI. Use CSS Colors. |
| GUI\_DASHBOARD\_REFRESH\_RATE | GUI | INT | N | N | N |  | Interval between automatic refreshes of GUI dashboard in seconds. If not specified GUI\_REFRESH\_RATE\*10 is used instead of it. |
| GUI\_REFRESH\_RATE | GUI | INT | N | N | Y | 5 | Interval between automatic refreshes of GUI header part in seconds |
| INITIALIZATION\_BEGIN | SCHEDULER | INT | Y | N | Y |  | Timestamp of last initialization begin. |
| INITIALIZATION\_CURRDATE\_RELATED | SCHEDULER | INT | Y | N | Y |  | Distinguisher if LOAD\_DATE value is related to current calendar date (1) or previous LOAD\_DATE value (0). |
| INITIALIZATION\_CURRDATE\_RELATED\_ROUND\_UP | SHEDULER | INT | Y | N | N | 0 | Distinguisher if LOAD\_DATE related to current timestamp is round down or up according to retention period. If not specified value is rounded down. |
| INITIALIZATION\_DELAY\_DAYS | SCHEDULER | INT | Y | N | Y | 0 | Number of days of processing delay behind current date. Zero value means one day delay (D-1). Used by framework checker for initialization issue detection. |
| INITIALIZATION\_DURATION\_MINUTES | SCHEDULER | INT | Y | N | Y | 20 | Initialization running duration threshold in minutes. If running duration value is above, alert is raised by Framework checker. |
| INITIALIZATION\_END | SCHEDULER | INT | Y | N | Y |  | Timestamp of last initialization end. |
| INITIALIZATION\_HOUR | SCHEDULER | INT | Y | N | Y |  | Hour when Initialization is launched. Used by framework checker for initialization issue detection. |
| INITIALIZATION\_IS\_RUNNING | SCHEDULER | INT | Y | N | Y |  | Initialization running indicator. Is running (1). It protects against start of another initialization before the current is finished. |
| INITIALIZATION\_MUST\_RUN | SCHEDULER | INT | Y | N | Y |  | If parameter is set to (1), initialization is done only if previous processing is finished. If parameter is set to (0) new LOAD\_DATE calculation is always done, other steps of initialization are done only if previous processing is finished. |
| INITIALIZATION\_PROBLEM\_COUNT | CHECKER | INT | N | N | N | 1 | Initialization issue event counter threshold. Alert is raised only If counter value is higher than threshold. |
| INITIALIZATION\_RETENTION\_PERIOD | SCHEDULER | INT | Y | N | Y |  | Retention period in minutes. Value is used for new LOAD\_DATE calculation from old LOAD\_DATE value. Use 1440 for daily processing. |
| LOAD\_DATE | SCHEDULER | INT | Y | N | Y |  | Processing date/timestamp. |
| LOAD\_SEQ\_NUM | SCHEDULER | INT | Y | N | Y |  | Processing sequence number. |
| MANUAL\_BATCH\_LOAD\_DATE | SCHEDULER | INT | Y | N | Y |  | Processing load date/timestamp used by manual batch. |
| MAX\_CONCURRENT\_JOBS | SCHEDULER | INT | Y | N | Y |  | Maximal number of concurrently running jobs used as number threshold of running jobs or temporal processing stop (0). |
| MAX\_CONCURRENT\_JOBS\_BCKP | SCHEDULER | INT | Y | N | Y |  | Temporal storage of MAX\_CONCURRENT\_JOBS value when Scheduler is stopped. |
| MAX\_CONCURRENT\_JOBS\_DFLT | SCHEDULER | INT | Y | N | Y |  | MAX\_CONCURRENT\_JOBS default value. |
| MAX\_CONCURRENT\_JOBS\_SET | SCHEDULER | INT | Y | N | Y |  | Maximal allowed value of MAX\_CONCURRENT\_JOBS parameter. |
| MAX\_LOAD\_DATE | SCHEDULER | INT | Y | N | Y |  | Maximal allowed value of LOAD\_DATE parameter. |
| MAX\_SKIPPED\_JOBS | SCHEDULER | INT | Y | N | N |  | Maximal number of skipped jobs during one engine cycle. If not specified MAX\_CONCURRENT\_JOBS value is used. |
| MESSAGE\_REITERATION\_COUNT | CHECKER | INT | N | N | Y | 1 | Maximal number of alert resent about one issue |
| MESSAGE\_REITERATION\_INTERVAL | CHECKER | INT | N | N | Y | 100 | Reiteration period in seconds for alert resent if issue persists. |
| NOTIFICATION\_DFLT | CHECKER | CHAR | N | N | Y | AVG\_DURATION | Monitored job execution attribute (AVG\_DURATION/AVG\_END\_TM). Used for job notification settings when new job is created. |
| NOTIFICATION\_ENABLED\_DFLT | CHECKER | INT | N | N | Y | 1 | Is job notification enabled. Used for job notification settings when new job is created. |
| PREV\_LOAD\_DATE | SCHEDULER | Date & Timestamp | Y | N | Y |  | LOAD\_DATE value of previous processing. Used by cleaning tasks etc. |
| SCHEDULER\_OFF\_COUNT | CHECKER | INT | N | N | N | 1 | Scheduler off issue counter threshold. Alert is raised only If counter value is higher than threshold. |
| SCHEDULER\_PROVIDED\_BY | SCHEDULER | INT | Y | N | Y |  | SCHEDULER - if standard processing is running or user name who invoked Manual batch. |
| SYSTEM\_OFF | SCHEDULER | INT | N | Y | Y |  | System bypass. System is eliminated from processing when value is set. (1) = system disabled, (0) system enabled |
| WATCHDOG\_INTERVAL | SCHEDULER | INT | N | N | Y | 300 | Threshold of Engine inactivity in seconds. If the value of inactivity is above threshold, Engine is considered to be dead and could be replaced by new Engine run. Engine inactivity is calculated as difference between current timestamp and ENGINE\_STATUS parameter value. |
| WATCHDOG\_INTERVAL\_FWRK | CHECKER | INT | N | N | Y | 1200 | Threshold of Engine inactivity in seconds. If the value of inactivity is above threshold, Engine dead alert will be raised by Framework checker. Engine inactivity is calculated as difference between current timestamp and ENGINE\_STATUS parameter value. |

## Framework checker tables

### CTRL\_NOTIFICATION table

**Job notification settings**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| JOB\_NAME | VARCHAR2(128) | N | Name of the job |
| NOTIFICATION\_ENABLED | NUMBER(38,0) | Y | Is notification enabled? |
| NOTIFICATION\_CD | NUMBER(38,0) | Y | The channel in which the notification will be processed |
| AVG\_DURARION\_TOLERANCE | NUMBER(38,0) | Y | Number of second passed before event  Job duration overrun is generated |
| AVG\_END\_TM\_TOLERANCE | NUMBER(38,0) | Y | Number of second passed before event Job end overrun is generated |
| CHECKED\_STATUS | NUMBER(38,0) | Y | Which job status has to be notified |
| MAX\_N\_RUN | NUMBER(38,0) | Y | Number of restarts for notification |
| ERROR\_CD | VARCHAR2(16) | Y | Error code for event sorting and recommendation generation |
| FINISHED\_AFTER\_INIT\_TOLERANCE | NUMBER(38,0) | Y | Number of minutes passed after init before event job not finish on time. |

Definition of job’s notification and also information which notification is applicable on job is stored in table. NULL value in column means that notification of this type is not required. Notification threshold is counted from:

1. Average duration calculated from statistics
2. Acceptable duration extension (parametr AVG\_DURATION\_FACTOR)
3. Acceptable duration offset (avg\_duration\_tolerance, avg\_end\_tm\_tolerance)

### CTRL\_NOTIFICATION\_TYPES table

**Notification type definition**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| NOTIFICATION\_TYPE\_CD | NUMBER(38,0) | N | Channel which is used for notification |
| NOTIFICATION\_TYPE\_DS | VARCHAR2(1024) | Y | Description of channel |

### CTRL\_NOTIFICATION\_RLTD table

**Transition table for notification type to channel mapping**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| NOTIFICATION\_CD | NUMBER(38,0) | N | Notification code |
| NOTIFICATION\_TYPE\_CD | NUMBER(38,0) | N | Channel used for notification |

### CTRL\_NOTIFICATION\_SEVERITY table

**Notification severity definition**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| ERROR\_CD | VARCHAR2(2048) | N | Error code |
| SEVERITY\_LEVEL\_CD | NUMBER(38,0) | Y | Severity level code |

## Statistics and events

### STAT\_STATUS table

|  |
| --- |
| **History table for job status trace**  When new initialization is done, content of SESS\_STATUS (it contains log from previous load) related to engine is copied into the STAT\_STATUS table. |

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| JOB\_ID | NUMBER(38,0) | N | Job identification |
| JOB\_NAME | VARCHAR2(128) | N | Name of the job |
| STREAM\_NAME | VARCHAR2(128) | N | Name of the stream |
| STATUS\_TS | TIMESTAMP(6)(11,6) | N | Timestamp of status change |
| LOAD\_DATE | DATE(7) | N | Load date |
| STATUS | NUMBER(38,0) | Y | Job status |
| N\_RUN | NUMBER(38,0) | N | Number of job launching |
| SIGNAL | VARCHAR2(16) | N | Signal which changed the status |
| APPLICATION\_ID | NUMBER(38,0) | Y | Application identification which drives the job |
| ENGINE\_ID | NUMBER(38,0) | N | Engine ID |
| SYSTEM\_NAME | VARCHAR2(128) | Y | System name where status change was applied |
| DWH\_DATE | DATE(7) | Y | DWH date |

### STAT\_JOB\_STATISTICS table

**History of job statistics**

It contains statistics overview of previous loads.

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| JOB\_NAME | VARCHAR2(128) | N | Name of the job |
| LOAD\_DATE | DATE(7) | Y | Load date |
| DAY\_IN\_WEEK | NUMBER(38,0) | Y | Day in week, Monday=1, Sunday=7 |
| DAY\_IN\_MONTH | NUMBER(38,0) | Y | Day in month, ultimo=999 |
| FIRST\_START\_TS | TIMESTAMP(6)(11,6) | Y | Timestamp when job was launched the first time |
| LAST\_START\_TS | TIMESTAMP(6)(11,6) | Y | Timestamp when job was launched the last time |
| LAST\_STATUS\_TS | TIMESTAMP(6)(11,6) | Y | Timestamp of last status change |
| END\_TS | TIMESTAMP(6)(11,6) | Y | Timestamp when job finished |
| LAST\_STATUS | NUMBER(38,0) | Y | Last status of the job |
| N\_RUN | NUMBER(38,0) | Y | Number of job launch |
| AVG\_DURATION | NUMBER(38,0) | Y | Average duration |
| AVG\_END\_TM | NUMBER(38,0) | Y | Average time when job finished |
| ENGINE\_ID | NUMBER(38,0) | Y | Engine identification |
| IGNORE\_STAT | NUMBER(38,0) | Y | If statistics is used for counting, ignore\_stat=0 |
| DWH\_DATE | DATE(7) | Y | DWH date |

### SESS\_JOB\_STATISTICS table

**Current job statistics**

Table is populated from SESS\_STATUS table during initialization and then table content is moved into STAT\_JOB\_STATISTICS table and finally table is refilled from averages from STAT\_JOB\_STATISTICS.

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| JOB\_NAME | VARCHAR2(128) | N | Name of the job |
| LOAD\_DATE | DATE(7) | Y | Load date |
| DAY\_IN\_WEEK | NUMBER(38,0) | Y | Day in week, Monday=1, Sunday=7 |
| DAY\_IN\_MONTH | NUMBER(38,0) | Y | Day in month, ultimo=999 |
| FIRST\_START\_TS | TIMESTAMP(6)(11,6) | Y | Timestamp when job was launched the first time |
| LAST\_START\_TS | TIMESTAMP(6)(11,6) | Y | Timestamp when job was launched the last time |
| LAST\_STATUS\_TS | TIMESTAMP(6)(11,6) | Y | Timestamp of last status change |
| END\_TS | TIMESTAMP(6)(11,6) | Y | Timestamp when job finished |
| LAST\_STATUS | NUMBER(38,0) | Y | Last status of the job |
| N\_RUN | NUMBER(38,0) | Y | Number of job launch |
| AVG\_DURATION | NUMBER(38,0) | Y | Average duration |
| AVG\_END\_TM | NUMBER(38,0) | Y | Average time when job finished |
| ENGINE\_ID | NUMBER(38,0) | Y | Engine identification |
| IGNORE\_STAT | NUMBER(38,0) | Y | If statistics is used for counting, ignore\_stat=0 |

### STAT\_LOG\_EVENT\_HIST table

**Events found by Framework checker**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| LOG\_EVENT\_ID | NUMBER(38,0) | N | Log event id |
| EVENT\_TS | TIMESTAMP(8)(11,8) | Y | Event timestamp |
| NOTIFICATION\_CD | NUMBER(38,0) | Y | Notification code |
| LOAD\_DATE | DATE(7) | Y | Load date |
| JOB\_NAME | VARCHAR2(2048) | N | Name of the job |
| JOB\_ID | NUMBER(38,0) | N | Job identification |
| SEVERITY\_LEVEL\_CD | NUMBER(38,0) | Y | Severity level code |
| ERROR\_CD | VARCHAR2(2048) | Y | Error code |
| EVENT\_CD | NUMBER(38,0) | Y | Error code |
| EVENT\_DS | VARCHAR2(2048) | Y | Event description |
| START\_TS | TIMESTAMP(8)(11,8) | Y |  |
| END\_TS | TIMESTAMP(8)(11,8) | Y |  |
| TRACKING\_DURATION | NUMBER(22) | Y | Job toughness |
| LAST\_STATUS | TIMESTAMP(8)(11,8) | Y | Last status of the job |
| N\_RUN | NUMBER(38,0) | Y | Actual number of job launch |
| CHECKED\_STATUS | NUMBER(38,0) | Y | Which job status has to be notified |
| MAX\_N\_RUN | NUMBER(38,0) | Y | Number of restarts for notification |
| AVG\_DURARION\_TOLERANCE | NUMBER(38,0) | Y | Number of second passed before event  Job duration overrun is generated |
| AVG\_END\_TM\_TOLERANCE | NUMBER(38,0) | Y | Number of second passed before event  Job end overrun is generated |
| ACTUAL\_VALUE | NUMBER(38,0) | Y | Actual value of checked attirbute |
| THRESHOLD | NUMBER(38,0) | Y | min. num of jobs when toughness reservation is applied |
| OBJECT\_NAME | VARCHAR2(2048) | Y | Object name |
| NOTE | VARCHAR2(2048) | Y | Note |
| SENT\_TS | TIMESTAMP(8)(11,8) | Y | Message sending timestamp |
| DWH\_DATE | DATE(7) | N | DWH date |
| ENGINE\_ID | NUMBER(38,0) | Y | Engine identification |
| RECOMMENDATION\_DS | VARCHAR2(2048) | Y | Recommendattion description |
| SYSTEM\_NAME | VARCHAR2(128) | Y | System which processed the job |

### STAT\_LOG\_MESSAGE\_HIST table

**History of sent messages**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| LOG\_EVENT\_ID | NUMBER(38,0) | N | Log event id |
| ERROR\_CD | VARCHAR2(2048) | N | Error code |
| ENGINE\_NAME | VARCHAR2(2048) | Y | Engine name |
| JOB\_NAME | VARCHAR2(2048) | N | Name of the job |
| JOB\_ID | NUMBER(38,0) | N | Job identification |
| SEVERITY | VARCHAR2(2048) | N | Severity |
| NOTIFICATION\_TYPE\_CD | NUMBER(38,0) | N | Channel used for notification |
| EVENT\_DS | VARCHAR2(2048) | N | Event description |
| RECOMMENDATION\_DS | VARCHAR2(2048) | Y | Recommendattion description |
| NOTE | VARCHAR2(2048) | Y | Note |
| ADDRESS | VARCHAR2(2048) | Y |  |
| DETECTED\_TS | TIMESTAMP(6)(11,6) | N | Timetamp when issue was identified |
| SENT\_TS | TIMESTAMP(6)(11,6) | Y | Message sending timestamp |
| SYSTEM\_NAME | VARCHAR2(128) | Y | System which processed the job |

## Lookups

### LKP\_APPLICATION table

**Lookup of application for job processing**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| APPLICATION\_ID | NUMBER(38,0) | Y | Application identification |
| IGNORE\_STATS | NUMBER(38,0) | Y | If ignore statistics if job was processed by this application |
| DESCRIPTION | VARCHAR2(1024) | Y | Description of application |
| ENGINE\_ID | NUMBER(38,0) | Y | Engine ID |
| IS\_ACTIVE | NUMBER(38,0) | Y | Activity indicator |

### LKP\_COUNTRY table

**Lookup of country**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| COUNTRY\_CD | VARCHAR2(4) | N | Country code |
| COUNTRY\_NAME | VARCHAR2(128) | N | Country name |
| DESCRIPTION | VARCHAR2(1024) | Y | Description |

### LKP\_ERROR\_CD table

**Lookup of error code**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| ERROR\_CD | VARCHAR2(16) | Y | Error code |
| ERROR\_CD\_NAME | VARCHAR2(36) | N | Error code name |
| ERROR\_CD\_NAME\_SHORT | VARCHAR2(7) | N | Error code name short |
| ERROR\_CD\_DESC | VARCHAR2(256) | N | Error code description |
| ERROR\_CD\_SOURCE | VARCHAR2(256) | Y | Error code source |
| ERROR\_CD\_RECOM | VARCHAR2(1024) | N | Error code reccomendation |

### LKP\_JOB\_CATEGORY table

**Lookup of job category**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| JOB\_CATEGORY | VARCHAR2(32) | N | Job subcategory name for parallelism control |
| DESCRIPTION | VARCHAR2(1024) | Y | Description of subcategory |
| ABORTABLE | NUMBER(22) | Y | **Is abort supported for the job category?** |

### LKP\_JOB\_TYPE table

**Lookup of job type**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| JOB\_TYPE | VARCHAR2(32) | N | Kind of job |
| DESCRIPTION | VARCHAR2(1024) | Y | Description of job type |

### LKP\_PHASE table

**Lookup of phase name**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| JOB\_PHASE | VARCHAR2(32) | N | Phase name |
| DESCRIPTION | VARCHAR2(1024) | Y | Description of phase name |

### LKP\_PLAN table

**Lookup of run plan**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| RUNPLAN | CHAR(9) | N | Run plan |
| DESCRIPTION | VARCHAR2(1024) | Y | Description of runplan |

## GUI tables

### GUI\_CHANGE\_MANAGEMENT table

**Change management**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| LABEL\_NAME | VARCHAR2(30) | N | Label name |
| LABEL\_STATUS | VARCHAR2(16) | N | Label status - Open or Close |
| USER\_NAME | VARCHAR2(128) | N | User name who created the label |
| CREATE\_TS | TIMESTAMP(6)(11,6) | N | Timestamp when label was created |
| DESCRIPTION | VARCHAR2(1024) | Y | Description what is label established for |
| ENV | CHAR(4) | N | Environment name where is label opened, DEVL in 99.99 percent |

### GUI\_CHANGE\_CONTROL table

**List of changes done via change management**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| LABEL\_NAME | VARCHAR2(30) | N | Label name |
| USER\_NAME | VARCHAR2(128) | N | User name who did the action |
| JOB\_NAME | VARCHAR2(128) | Y | Name of the job |
| UID\_INDICATOR | CHAR(1) | Y | Operatin indicator M - merge, D - delete |
| CMD\_TS | TIMESTAMP(6)(11,6) | N | Execution timestamp |
| SEQ\_NUM | NUMBER(22) | N | Sequence number |
| CMD | VARCHAR2(4000) | N | SQL command |

### GUI\_LOG\_CTRL\_ACTION table

**Log of GUI control actions**

|  |  |  |  |
| --- | --- | --- | --- |
| COLUMN\_NAME | DATATYPE | NULLABLE | COMMENT |
| USER\_NAME | VARCHAR2(128) | N | User name who did the action |
| ACTION | VARCHAR2(128) | N | Action identification |
| ACTION\_TS | TIMESTAMP(6)(11,6) | Y | Timestamp when action was done |
| SQL\_CODE | VARCHAR2(4000) | Y | SQL code of action |
| DWH\_DATE | DATE(7) | N | DWH date |