DPP Utility User's Guide v24.3

Date: 02/10/2024
Author: Jean-Marc MALMEDY
Document version: 24.3

1. Table of content

2. Introduction

This document is a technical description of the DPP Utility and a complete user guide with several use cases.

2.1. Overview

The **PL/SQL Data Pump Utility**, or **DPP Utility** in this document, is a PL/SQL utility that aims at streamlining the data pump operations in Oracle database.

The DPP Utility is a convenient solution to replicate Oracle database schemas between several instances. It is fully configurable and totally managed using a single PL/SQL package.

2.2. Main functionalities

The main functionalities of the utility are:

1. Defining reusable configurations to export, transfer and import database schema dumps.
2. Exporting a database schema in a data pump dump file using pre-defined configurations or specific parameters.
3. Transferring a data pump dump file from a database instance to another one using pre-defined configurations or specific parameters.
4. Importing a data pump dump file into a database schema using pre-defined configurations or specific parameters.
5. Directly importing a database schema from a database instance to another one through a network link using pre-defined configurations or specific parameters.
6. Timeout monitoring and detection.

2.3. Assumptions

In this document it is assumed that they are two database environments:

- **PROD**: the production environment
- **TEST**: the test environment

Usually, in a classical use case, the **PROD** environment is the source of the data pump flow while the **TEST** environment is the target one.

2.4. Differences with regular Data Pump

The DPP Utility is based on the regular Oracle Data Pump utility. However, the DPP Utility proposes some enhancements and additional functionalities compared to the Data Pump utility:

- Transfer of the dump files from the source system to the target system.
- Automatic execution of pre and post actions before or after export and import operations.
- Ability to kill sessions and lock schemas during export and import operations.
- Easy to deploy bundle.
- Adapted for AWS and COP environments.
- Automatically sending a mail at the end of an export, import or transfer operation. Mail sending is compatible with AWS SES (Simple Email Service).
- Job monitoring and timeout detections with mail sending.

3. Getting started

3.1. Installation

The detailed installation guide is described in the installation chapter.

The DPP Utility needs to be installed in each database instance where data pump operations must be performed. It may be installed only once in a dedicated schema, there is no need to install it in all schemas that need to be managed by the DPP Utility. This database schema must be created prior the installation and must fulfill some conditions described in the Installation / APP_DPP_X schema chapter. All schemas that need to be managed by the DPP Utility must also be granted with some privileges described in the Client application schema chapter.

The installation kit is available as a DBM Utility bundle. No specific parameter needs to be initialized for the installation of the tool.

3.2. Operations

3.2.1. Exporting dump file

To generate a dump file of a database schema, the following operations must be performed:

1. Setting up the export parameters in the following tables as described in the set up chapter:
   - instances
   - parameters
   - roles
   - schemas
   - actions
   - recipients
   - schema options
2. Running the export: `EXEC dpp_job_krn.export_schema('schema_name');`
3. Getting the dump file(s)

3.2.2. Importing dump file

To import a dump file in a database schema, the following operations must be performed:

1. Setting up the import parameters in the following tables as described in the set up chapter:
   - instances
   - parameters
   - roles
   - schemas
   - actions
   - recipients
   - schema options
   - schema relationships
   - protected objects
2. Running the import: `EXEC dpp_job_krn.import_schema('source_schema_name', 'target_schema_name');`

3.2.3. Transferring dump file

To transfer a dump file from the source environment to the target one, the following operations must be performed:

1. Setting up the transfer parameters in the following tables as described in the set up chapter:
   - instances
   - parameters

- roles
- schemas
- recipients
- schema options

2. Running the transfer: `EXEC transfer_dumpfiles('func_name', 'db_link_name');`

3.2.4. Timeout monitoring

A monitoring can be set up for each schema and type of job in order to detect the jobs that possibly could not terminate correctly. This is done by detecting the timeouts as described in the Timeout monitoring chapter. The monitoring is configured in the following tables:

- schema options
- recipients
- parameters

Once the parameters are defined, a job is executed every five minutes to detect the jobs in timeout. The timeout detection does not have to be executed manually.

4. Installation

The DPP Utility needs to be installed in all database instances where some data pump operations need to be performed. It is not needed to install the utility in all schemas where import and/or export operations must be performed. Installing the tool in one single schema per database instance is enough.

The DPP Utility components are mainly installed in a dedicated schema called **APP_DPP_X** where **X** stands for the environment (**D** for development, **T** for test, **P** for production, ...). Depending on the environment (Data Center, Amazon Web Services or Cloud on Premise), some components can be installed in other schemas too as described below.

The DPP Utility is available as an installation bundle packaged with the **DBM Utility** which is the new tool developed by the DBE CoE to manage the installation, upgrade and uninstallation of its PL/SQL utilities.

4.1. Pre-requisites

Depending on the version of the DPP Utility that needs to be installed, the pre-requisites are slightly different:

- From version 23.0 to 24.0: the MAIL Utility is required and must be installed before installing the DPP Utility.
- Starting from version 24.1:
  - At least the version 24.0 of the MAIL Utility is required and must be installed before installing the DPP Utility.
  - The SEC Utility is required and must be installed before installing the DPP Utility.

Both the MAIL Utility and the SEC Utility are DBE CoE PL/SQL utilities that must be installed using the DBM Utility. The dependencies are fully managed by the DBM Utility.

4.2. Installation kit

The DPP Utility is part of the PL/SQL toolkit and can be installed using the DBM tool.

The following directories are part of the installation kit:

- **DOC:** The documentation of the DPP Utility.
- **releases:** This directory contains the installation material with a sub-directory for each release.
  - **ALL:** This directory contains the material common to all versions of the DPP Utility.
    - **config:** This directory contains a configuration file (**dpp_utility.conf**) that stores the parameters common to all versions.
    - **precheck:** This directory contains a script (**precheck.sql**) that checks several parameters and conditions that must be fulfilled before installing the DPP Utility.
    - **uninstall:** This directory contains some scripts that uninstall the DPP Utility.
  - **XX.X:** A sub-directory for each version of the DPP Utility where **XX.X** must be replaced with the version number.
    - **config:** This directory contains configuration and parameter files specific to the current version of the tool.
    - **install:** This directory contains the scripts that install the current version of the tool.
    - **upgrade:** This optional directory contains the scripts that migrate the tool from the previous version to the current one.

Actually, there is no need to know those directories since the installation is automated by the DBM Utility.

4.3. Base parameters

Some parameter values need to be defined before starting the installation. These parameters are stored in the following file in the installation kit:
ec_plsql_toolkit\apps\30_dpp_utility\releases\ALL\config\dpp_utility.conf

The parameters can also be manually initialized using the following DBM tool command: `@dbm-cli configure dpp`

Actually, all of the parameters defined in this file should keep their default values. So except for very specific setup, the default parameter values should not be modified before installing.

4.4. Database schemas and environments

Two kinds of database schemas are part of a classical DPP Utility architecture:

- **installation / APP_DPP_X:** This is the database schema the DPP Utility is installed in. It stores the configuration tables of the utility and the corresponding PL/SQL code in several packages. According to the usual naming conventions, such a schema is supposed to be called **APP_DPP_X** where **X** is replaced with a letter indicating the current environment (**D** for Development, **T** for Test, **P** for Production, ...). Those are pure naming conventions in use at the European Commission, there are no technical constraints about the schema name.
- **client application schemas:** Those are the database schemas that need to be managed by the DPP Utility, for data pump and transfer operations.

Each kind of database schema requires specific configurations that are explained further in this document.

The database instances at the European Commission can be created in three different kinds of environment:

- **Data Center:** Those are database instances hosted in the infrastructures of European Commission's Data Center.
- **AWS - Amazon Web Service:** Those are database instances hosted in Amazon Web Service's infrastructure.
- **COP - Cloud On Premise:** Those are cloud-based database instances but fully managed as a service by the Data Center.

Each of those environments requires also a specific configuration.

That means that the installation procedure of the DPP Utility must deploy the right configuration depending on the kind of schema and on the kind of environment. Fortunately, the DBM Utility bundle can detect them and automatically deploy the right configuration.

4.4.1. Installation / APP_DPP_X schema

A dedicated schema needs to be created for the deployment of the DPP Utility. It stores the configuration tables and the PL/SQL code. There are no technical constraints about the naming of this schema but it is recommended to apply the usual naming conventions. So a suitable schema name for a deployment in development environment at the European Commission could be: **APP_DPP_D**.

Depending on the kind of environment, this schema user must be granted with several roles and privileges.

4.4.1.1. Data Center database

According to the usual naming conventions in use at the European Commission, the following table spaces should be used for the DPP Utility schema in a Data Center database:

Table space name	Use
DPPTAB	for the data
DPPIDX	for the indexes

The schema user must be granted with the following roles:

Role name
CONNECT
DC_RESOURCE
JAVAUSERPRIV
SELECT_CATALOG_ROLE
AQ_ADMINISTRATOR_ROLE

When the database is hosted at the Data Center, they are responsible for the database administration operations. Therefore the **APP_DPP_X** user does not need to be granted with DBA privileges. The few administration operations that are permitted are executed using dedicated stored procedure developed by the Data Center. The user must be granted with the execution privilege on those procedures:

Procedure name
c##ops$oracle.dc_dba_mgmt_kill_sess_dedic_db
c##ops$oracle.dc_dba_mgmt_lock_user

The schema must also be granted with the following system privileges:

Privilege name
ALTER SYSTEM
ALTER USER
ADVISOR
ALTER DATABASE LINK
CREATE ANY CONTEXT
CREATE ANY PROCEDURE
CREATE DATABASE LINK
CREATE JOB
CREATE PROCEDURE
CREATE SEQUENCE
CREATE SESSION
CREATE SYNONYM
CREATE TABLE
CREATE TRIGGER
CREATE VIEW
DEBUG CONNECT SESSION
DEQUEUE ANY QUEUE
DROP ANY CLUSTER
DROP ANY CONTEXT
DROP ANY DIMENSION
DROP ANY INDEX
DROP ANY MATERIALIZED VIEW
DROP ANY OPERATOR
DROP ANY PROCEDURE
DROP ANY SEQUENCE
DROP ANY SQL PROFILE
DROP ANY SYNONYM
DROP ANY TABLE
DROP ANY TRIGGER
DROP ANY TYPE
DROP ANY VIEW
ENQUEUE ANY QUEUE
EXECUTE ANY PROCEDURE
MANAGE ANY QUEUE
MANAGE SCHEDULER
RESTRICTED SESSION
SELECT ANY DICTIONARY
SELECT ANY TABLE
UNLIMITED QUOTAS on both user tablespaces for Data and indexes (example DPPTAB and DPPIDX tablespaces)

The schema must also be granted with the following object privileges:

Object privilege
SELECT on V$PARAMETER
SELECT on V$INSTANCE
SELECT on V$LOCKED_OBJECT
SELECT on V$SESSION
SELECT on DBA_USERS
SELECT on DBA_DIRECTORIES
SELECT on DBA_JOBS_RUNNING
SELECT on DBA_JOBS
SELECT on DBA_OBJECTS
EXECUTE ON DC_DBA_MGMT_KILL_SESS_DEDIC_DB
EXECUTE ON DC_DBA_MGMT_LOCK_USER
EXECUTE on SYS.DBMS_LOCK
EXECUTE on SYS.DBMS_FILE_TRANSFER
EXECUTE ON SYS.DBMS_AQ (if there are Oracle Queues)
EXECUTE ON SYS.DBMS_AQADM (if there are Oracle Queues)
READ and WRITE in the oracle directories.
SELECT on SYS.OBJ$
SELECT ON DBA_SERVICES (used to determine if we are in DC or AWS)

The **SELECT ANY DICTIONARY** privilege can replace both the last ones.

If the user is not granted with those privileges, it must be asked to the Data Center to grant them.

4.4.1.2. AWS - Amazon Oracle RDS instance

According to the usual naming conventions in use at the European Commission, the following table spaces should be used for the DPP Utility schema in an AWS database:

Table space name	Use
DPPTAB	for the data
DPPIDX	for the indexes

The schema user must be granted with the following roles:

Role name
CONNECT
CTXAPP
DBA
RESOURCE
JAVAUSERPRIV
RDS_JAVA_ADMIN
SELECT_CATALOG_ROLE
RECOVERY_CATALOG_OWNER
SCHEDULER_ADMIN
AQ_ADMINISTRATOR_ROLE
AQ_USER_ROLE
EXECUTE_CATALOG_ROLE
EXP_FULL_DATABASE
GATHER_SYSTEM_STATISTICS
HS_ADMIN_EXECUTE_ROLE
HS_ADMIN_SELECT_ROLE
IMP_FULL_DATABASE
OEM_ADVISOR
OEM_MONITOR
XDB_SET_INVOKER
XDBADMIN

The schema user must be granted with the following system privileges:

System privilege
ADMINISTER DATABASE TRIGGER
ADVISOR
ALTER DATABASE LINK
ALTER PUBLIC DATABASE LINK
ALTER USER
CHANGE NOTIFICATION
CREATE ANY CONTEXT

System privilege
CREATE ANY PROCEDURE
CREATE DATABASE LINK
CREATE JOB
CREATE PROCEDURE
CREATE SEQUENCE
CREATE SESSION
CREATE SYNONYM
CREATE TABLE
CREATE TRIGGER
CREATE VIEW
DEBUG CONNECT SESSION
DEQUEUE ANY QUEUE
DROP ANY CLUSTER
DROP ANY CONTEXT
DROP ANY DIMENSION
DROP ANY INDEX
DROP ANY INDEXTYPE
DROP ANY LIBRARY
DROP ANY MATERIALIZED VIEW
DROP ANY OPERATOR
DROP ANY PROCEDURE
DROP ANY SEQUENCE
DROP ANY SQL PROFILE
DROP ANY SYNONYM
DROP ANY TABLE
DROP ANY TRIGGER
DROP ANY TYPE
DROP ANY VIEW
ENQUEUE ANY QUEUE
EXECUTE ANY PROCEDURE
EXEMPT ACCESS POLICY
EXEMPT IDENTITY POLICY
EXEMPT REDACTION POLICY
FLASHBACK ANY TABLE
GRANT ANY OBJECT PRIVILEGE
MANAGE ANY QUEUE
MANAGE SCHEDULER
RESTRICTED SESSION
SELECT ANY DICTIONARY
SELECT ANY TABLE
UNLIMITED QUOTAS on both user tablespaces for Data and indexes (example DPPTAB and DPPIDX Tablespaces)

The schema user must also be granted with the following object privileges:

Object privilege
EXECUTE on SYS.DBMS_LOCK
EXECUTE on SYS.DBMS_FILE_TRANSFER
SELECT on SYS.OBJ$
EXECUTE ON SYS.DBMS_AQ (if there are Oracle Queues)
EXECUTE ON SYS.DBMS_AQADM (if there are Oracle Queues)
READ and WRITE in the oracle directories.
EXECUTE on UTL_MAIL, UTL_RAW, UTL_TCP and UTL_SMTP
EXECUTE on rdsadmin.rdsadmin_util

Note that for the three last ones, you need to call `rdsadmin.rdsadmin_util.grant_sys_object` . The operation cannot be performed by classic `GRANT EXECUTE` DDL statement.
Example: `exec rdsadmin.rdsadmin_util.grant_sys_object('UTL_MAIL', 'APP_DPP_D', 'EXECUTE')`

In the AWS environment, the functionalities of the Data Center procedures are not available. They are replaced with a dedicated procedure that is part of the DPP Utility deployment kit:
dc_dba_mgmt_lock_user
No special privileges must then be requested.

4.4.1.3. COP - Cloud On Premise

According to the usual naming conventions in use at the European Commission, the following table spaces should be used for the DPP Utility schema in a COP database:

Table space name	Use
DPPTAB	for the data
DPPIDX	for the indexes

The schema user should be granted with the following roles:

Role name
CONNECT
DC_RESOURCE
JAVAUSERPRIV
SELECT_CATALOG_ROLE
AQ_ADMINISTRATOR_ROLE

The schema user should be granted with the following system privileges:

System privilege
ADMINISTER DATABASE TRIGGER
ADVISOR
ALTER DATABASE LINK
ALTER PUBLIC DATABASE LINK
ALTER USER
CHANGE NOTIFICATION
CREATE ANY CONTEXT
CREATE ANY PROCEDURE
CREATE DATABASE LINK
CREATE JOB
CREATE PROCEDURE
CREATE SEQUENCE
CREATE SESSION
CREATE SYNONYM
CREATE TABLE
CREATE TRIGGER
CREATE VIEW
DEBUG CONNECT SESSION
DEQUEUE ANY QUEUE
DROP ANY CLUSTER
DROP ANY CONTEXT
DROP ANY DIMENSION
DROP ANY INDEX
DROP ANY INDEXTYPE
DROP ANY LIBRARY
DROP ANY MATERIALIZED VIEW
DROP ANY OPERATOR
DROP ANY PROCEDURE
DROP ANY SEQUENCE
DROP ANY SQL PROFILE
DROP ANY SYNONYM
DROP ANY TABLE
DROP ANY TRIGGER
DROP ANY TYPE
DROP ANY VIEW
ENQUEUE ANY QUEUE
EXECUTE ANY PROCEDURE
EXEMPT ACCESS POLICY
EXEMPT IDENTITY POLICY
EXEMPT REDACTION POLICY
FLASHBACK ANY TABLE
GRANT ANY OBJECT PRIVILEGE
MANAGE ANY QUEUE
MANAGE SCHEDULER
RESTRICTED SESSION
SELECT ANY DICTIONARY
SELECT ANY TABLE

System privilege
UNLIMITED QUOTAS on both user tablespaces for Data and indexes (example DPPTAB and DPPIDX tablespaces)

The schema user should be granted with the following object privileges:

Object privilege
EXECUTE ON SYS.DBMS_AQ
EXECUTE ON SYS.DBMS_AQADM
EXECUTE on SYS.DBMS_FILE_TRANSFER
EXECUTE on SYS.DBMS_LOCK
EXECUTE on SYS.DBMS_SCHEDULER
EXECUTE ON SYS.UTL_MAIL
READ and WRITE in the oracle directories

In the COP environment, the functionalities of the Data Center procedures are not available. They are replaced with two dedicated procedures that are part of the DPP Utility deployment kit: **dc_dba_mgmt_kill_sess_dedic_db** and **dc_dba_mgmt_lock_user**.
No special privileges must then be requested.

4.4.2. Client application schema

The schema user must be granted with the same privileges independently from the deployment environment. Those needed privileges are the same in a Data Center instance, an AWS instance or a COP instance.

The user must be granted with the following system privileges:

System privilege
CREATE JOB
CREATE TABLE
CREATE TRIGGER
CREATE VIEW
And any other system privilege which is covered by RESOURCE, since they will not be resolved by the PL/SQL utility.

The schema user must be granted with the following object privileges:

Object privilege
READ and WRITE in the oracle directories
SELECT on SYS.OBJ$
SELECT on DBA_SERVICES (needed to determine whether we are on DC premises, AWS or COP)

The **SELECT ANY DICTIONARY** privilege can replace both the last ones.

4.4.2.1. Source schema in case of import through a network link

When an import job is performed through a network link the source schema needs to be granted with the following privileges which slightly differ depending on the environment.

4.4.2.1.1. For an import from a Data Center database

Those are the requested privileges:

- No particular system privileges.
- The **SELECT** privilege on the **sys.obj$** view.
- The **SELECT_CATALOG_ROLE** role.

4.4.2.1.2. For an import from a AWS database

Those are the requested privileges:

- No particular system privileges.
- The **SELECT** privilege on the **sys.obj$** view.
- The **EXP_FULL_DATABASE** role. Usually, the documentation mentions that the **IMP_FULL_DATABASE** role needs to be granted too. This is because both those roles are usually granted together.

In some cases, forcing the default role of the source schema is needed. This is achieved by executing the following command as administrator:

```
ALTER USER source_schema
DEFAULT ROLE imp_full_database,exp_full_database;
```

Where **source_schema** must be replaced with the source schema name.

4.4.2.1.3. For an import from a Cloud On Premise database

Those are the requested privileges:

- The **SELECT** privilege on the **sys.obj$** view. If this privilege cannot be granted, it can be replaced with the **SELECT ANY DICTIONARY** system privilege.
- The **DATAPUMP_EXP_FULL_DATABASE** role.
- The **SELECT_CATALOG_ROLE** role.

According to the Oracle documentation, the source schema user must be granted with the **DATAPUMP_EXP_FULL_DATABASE** role only if the target schema user is granted with the **DATAPUMP_IMP_FULL_DATABASE** role.

4.5. Deployment

The following DBM Utility commands can be used for the deployment of the DPP Utility:

- The DPP Utility is installed with the following command: `@dbm-cli install dpp_utility`
- The DPP Utility is upgraded to the last version with the following command: `@dbmcli migrate dpp_utility` .
  If no previous version of the DPP Utility was installed when running the command, it works like the **install** command.
- The DPP Utility is uninstalled with the following command: `@dbm-cli uninstall dpp_utility`
- The DPP Utility installation is validated with the following command: `@dbm-cli validate dpp_utility`

4.6. Exposing tool

Once the tool is installed in its own database schema, it can be made available to all other database users and schemas or to specific ones. This is actually not a functionality of the DPP Utility itself but one of the DBM Utility.

The tool is made available to all users and schemas by granting all users (**PUBLIC**) with access to the application tables and packages and by creating public synonyms for those objects. To make the tool available to a specific user, this user must be granted with access to the application tables and packages and synonyms for those objects must be created in this client schema. Granting access and creating the synonyms is done by the DBM Utility.

The following DBM Utility commands can be used to expose the tool to other users:

- The DPP Utility is exposed to all users or is made public with the following command: `@dbm-cli expose dpp_utility public` .
- The DPP Utility is exposed to a specific user or made available for this user with the following command: `@dbm-cli expose dpp_utility {user}` where **{user}** must be replaced with the name of the user the tool must be exposed to.

5. Operations

5.1. Exporting dump file

A common use case of the DPP Utility is the generation of a Data Pump dump file.

5.1.1. Dump file storage

The Data Pump dump file generated by the DPP Utility is stored in an Oracle directory the current user has read and write access to. Oracle stores the dump-files on the server-side, this means that the UNIX user running Oracle must have read/write privileges on said directory.

This directory may have different characteristics depending on the current environment:

- **DC - Data Center database:** The test and production environments may have different storage locations.
- **AWS - Amazon Web Services database:** The export directory is the one configured by Amazon, usually **DATA_PUMP_DIR**.
- **COP - Cloud On Premise database:** According to the naming conventions in use at the European Commission expressed in the Oracle COP user guide, the export directory is usually **DP_DIR_S3**.

Still depending on the environment, this is the usual matching between the Oracle directory and the OS/Unix one:

Environment	Oracle directory	OS/Unix directory
Data Center	DBCC_DBIN	/ec/dev/server/oracle/digit-files/dbccd/dbin
Data Center	DBCC_DBOUT	/ec/dev/server/oracle/digit-files/dbccd/dbout
Amazon Web Services	DATA_PUMP_DIR	/rdsdbdata/datapump
Cloud On Premise	DP_DIR_S3	/oracle/dp

For a COP database, as described in the Oracle COP user guide, the Oracle directory is mapped on a S3 bucket mounted on the */oracle/dp* OS directory. The user guide explains how such an S3 bucket should be requested and mapped in the COP Oracle database.

The dump file follows some naming conventions. The dump file name is made of several concatenated parts:

- the name of the exported database schema
- the export date in **YYYYMMDD** format
- a three digits number indicating the number of the export on a same day
- a three digits number indicating the parallelism number; when several CPUs are used to generate the dump file, each of them produces a file identified with this parallelism number
- the **.exp** extension; during the export operation, the file has temporarily the **.bsy** extension

For example, the **APP_DBECOE_D20240517002002.exp** dump file name may be interpreted as follows:

- The exported schema is **APP_DBECOE_D**.
- The export date is the 17th of May 2024.
- This the third export of the day, the numbering starting at **000**.
- This is the second dump file of this export made with parallelism.

5.1.2. Setting up export

An DPP Utility export job is defined with several parameters that need to be inserted in several database tables.

5.1.2.1. DPP_INSTANCES - Database instance

The database instance must be defined in the **dpp_instances** table as follows:

Field	Description	Example value
ITE_NAME	database instance name	DBCC_DIGIT_01_D
DESCR_ENG	description en English	DBE CoE's development database
DESCR_FRA	description in French	base de données de développement du DBE CoE
PRODUCTION_FLAG	flag indicating whether this is a production instance (**Y** or **N**)	N
ENV_NAME	environment name (**DC**, **AWS** or **COP**)	DC
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

At least one instance must be defined.

5.1.2.2. DPP_PARAMETERS - General parameters

Several general parameters must be defined in the **dpp_parameters** table as follows:

Field	Description	Example value
PRR_NAME	parameter name	g_dpp_out_dir
PRR_VALUE	parameter value	DATA_PUMP_DIR
DESCR_ENG	description in English	Oracle directory for export
DESCR_FRA	description in French	répertoire Oracle d'export

Field	Description	Example value
ITE_NAME	The name of the database instance defined in **dpp_instance** if the parameter relates to a particular instance. If the parameter is general, this field may be left empty.	DBCC_DIGIT_01_D
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

The following parameters need to be defined for an export job:

Parameter	Mandatory	Comment
g_dpp_out_dir	yes	name of the Oracle directory the dump files must be generated in
smtp.dev_recipient	no	The DPP Utility can generate a mail at the end of the job indicating whether it was executed successfully or with error. If the job is not executed in production environment, the mail is not sent to the regular recipients to avoid spamming. If a mail address is mentioned for the parameter, the mail will be sent to this address instead of the regular recipients.
smtp.host	no	The SMTP server that needs to be used to send the mail at the end of the job. If it is not mentioned, the default (**localhost**) server is used.
smtp.port	No	This optional parameter stores the IP port of the SMTP server.
smtp.domain	No	This optional parameter stores the SMTP server domain.
smtp.username	No	If the SMTP server requires some authentication, the user ID is stored in this parameter.
smtp.wallet_path	No	Some SMTP servers need a certificate to complete the authentication. This parameter stores the path of the Oracle directory the **cwallet.sso** certificate file is stored in.
smtp.sender	No	This parameter stores the email address of the mail sender.
smtp.default_recipient	No	This is the default recipient a mail is sent to when the list of recipients cannot be determined. This is the case when the schema functional name given as parameter does not exist.

5.1.2.3. DPP_ROLES - Database role

If a specific database role must be used to execute the export job, it must be defined in the **dpp_roles** table as follows:

Field	Description	Example value
RLE_NAME	role name	EXPORT_ROLE
DESCR_ENG	description in English	export role
DESCR_FRA	description in French	role d'export
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

Defining a role is optional since they are only needed when some code must be executed via the **dpp_actions** configuration before or after an export or import operation. Such an action could be restoring some privileges to the role.

5.1.2.4. DPP_SCHEMAS - Schemas

The database schemas that need to be exported must be defined in the **dpp_schemas** table as follows:

Field	Description	Example value
SMA_ID	schema unique identifier	1
ITE_NAME	instance name as defined in the **dpp_instances** table	DBCC_DIGIT_01_D
RLE_NAME	optional role name as defined in the **dpp_roles** table	EXPORT_ROLE
STE_NAME	schema type as defined in the **dpp_schema_types** table	MAIN
FUNCTIONAL_NAME	an unique functional name given to the schema	SRCDEV
SMA_NAME	real name of the database schema	APP_DPP4SRC_D
PRODUCTION_FLAG	flag indicating whether the schema is a production one (**Y** or **N**)	N
DATE_FROM	start date of the schema validity	2024-05-22 13:07:34
DATE_TO	optional end date of the schema validity, the schema has no end date if omitted	2030-12-31 23:59:59
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

At least one schema must be defined in this table.

5.1.2.5. DPP_ACTIONS - Specific actions

Some actions must be executed before or after the export operations. They need to be defined in the **dpp_actions** table as follows:

Field	Description	Example value
SMA_ID	the schema ID as defined in the **dpp_schemas** table	1
ATN_USAGE	the flow direction (**I** for an import, **E** for an export, **T** for a transfer and **M** for a monitoring)	E
ATN_TYPE	prefix or postfix action (**PREFIX** or **POSTFIX**)	POSTFIX
EXECUTION_ORDER	execution order	1
BLOCK_TEXT	the code to be executed	BEGIN post_export(); END;
ACTIVE_FLAG	whether the action is active (**Y** or **N**)	Y

Field	Description	Example value
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

For each schema, several actions or none may be defined.

If the action code stored in **BLOCK_TEXT** contains some sensitive pieces of information, like passwords for example, they can be encrypted using the SEC Utility. The sensitive information can be replaced with **{DPPCRYPT:var}** where **var** needs to be replaced with the username or variable associated with the sensitive value. The value will be decrypted when the action will be executed.

5.1.2.6. DPP_RECIPIENTS - Mail recipients

At the end of the export process, the DPP Utility can send a mail to some recipients indicating whether the job was executed successfully or with some errors. The mail recipients are defined in the **dpp_recipients** table as follows:

Field	Description	Example value
SMA_ID	the schema ID as defined in the **dpp_schemas** table	1
EMAIL_ADDR	the email address	fname.lname@ec.europa.eu
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

No, one or several recipients may be defined for each schema. If none is defined, no mails will be sent at the end of the export process.

5.1.2.7. DPP_SCHEMA_OPTIONS - Schema options

Several options can be defined to fine tune the export job. Those options are more or less the ones of the regular Oracle Data Pump tool with some additional ones. Those options are defined in the **dpp_schema_options** table as follows:

Field	Description	Example value
SMA_ID	the schema ID as defined in the **dpp_schemas** table	1
OTN_NAME	option name	PARALLEL
STN_VALUE	option value	10
STN_USAGE	the flow direction (**I** for an import, **E** for an export, **T** for a transfer)	E
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

Several options can be defined for a schema and a flow. They are all defined in the **dpp_options** table and the possible values, when enumerated, are available in the **dpp_option_allowed_values** table. Those are the options that are available for an export job:

Option name	Description	Possible values
BLOCK	This option blocks any import or export operation. When scheduling background jobs it is not needed to kill the job to stop a export/import action. Just setting the option will skip any export/import.	**Y** for yes or **N** for no
EMAIL_RESULT	Emails the final result to the designated email list. If the job ended in **ERROR** the log from **dpp_job_logs** will be appended to the mail.	**Y** for yes or **N** for no
EXEC_PREFIX	The export job has now the possibility to execute scripts in the target schema, before exporting data.	**Y** for yes or **N** for no
METADATA_FILTER	Filter to exclude tables whose name is compliant with the given pattern.	NOT LIKE "xxx%" pattern. Example: NOT LIKE "TMP%"
PARALLEL	The number of parallel threads used for exporting data to dump files.	Any numeric value greater than ZERO (Default: uses the number of processors available to oracle) Example: PARALLEL=10
TIMEOUT_MONITORING	Activates or deactivates the timeout monitoring for the current schema.	**Y** for yes or **N** for no
TIMEOUT_DELAY	Number of minutes defining the delay after which the job is considered being in timeout.	Numeric value, number of minutes.
DATA_FILTER	Create a filter on a table in order to exclude some data. Such an entry must be created for each table that must be filtered. The Oracle documentation should be checked for more details.	The value is a concatenation of three values separated with the "#" character. These three values are the name of the table to be filtered (the filter is for all tables if omitted), the filter type (**INCLUDE_ROWS**, **PARTITION_EXPR**, **PARTITION_LIST**, **SAMPLE** or **SUBQUERY**) and the filter value depending on the filter type. If the table name is not mentioned, the separator must be present anyway. Example: TST_EMPLOYEE#SUBQUERY#WHERE EMP_ID < 400

Option name	Description	Possible values
COMPRESSION	Define the type of compression that must be applied on the dump file as described in the Oracle documentation.	**DATA_ONLY**, **METADATA_ONLY**, **ALL** or **NONE**
COMPRESSION_ALGORITHM	Define the algorithm to be used for data compression as described in the Oracle documentation.	**BASIC**, **LOW**, **MEDIUM** or **HIGH**
ENCRYPTION	Define the encryption method to be used for the data as described in the Oracle documentation	**ALL**, **DATA_ONLY**, **ENCRYPTED_COLUMNS_ONLY**, **METADATA_ONLY** or **NONE**
ENCRYPTION_MODE	Define the mode of encryption as described in the Oracle documentation.	**PASSWORD**, **TRANSPARENT** or **DUAL**
ENCRYPTION_PASSWORD	This is the encryption password when the **PASSWORD** encryption mode is selected.	any valid password
LOGTIME	Define the type of timestamps to be mentioned in the logs generated by the Data Pump as described in the Oracle documentation.	**NONE**, **STATUS**, **LOGFILE** or **ALL**
DATA_REMAP	With this option, a transformation can be applied to a column at export time as explained in the Oracle documentation. Such an entry must be created for each column that must be transformed.	The value is the concatenation of four fields separated with the "#" character. Those fields are the name of the remap (**COLUMN_FUNCTION** usually), the name of the table the column belongs to, the column name and the name of a PL/SQL function in a package that applies the transformation. The function must have a single parameter of the same type as the column to be transformed and must return a value of the same type. For example: COLUMN_FUNCTION#TST_EMPLOYEE#FIRST_NAME#DATA_REMAP_TEST.TO_UPPER

5.1.3. Preparing schema for export

Some actions may be required to prepare a schema for the export of the dump.

If there are some queues in use in the schema to be exported, the following code must be executed while logged with the user of the schema to be exported:

```
BEGIN
   -- needed only if there are Oracle Advanced Queues to manage
   DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('ENQUEUE_ANY', '&&APP_DPP',FALSE);
   DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('DEQUEUE_ANY', '&&APP_DPP',FALSE);
   DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('MANAGE_ANY', '&&APP_DPP',FALSE);
END;
/
```

5.1.4. Running the export

Once all needed parameters have been defined, the export job can be executed. It is implemented by the **export_schema** procedure of the **dpp_job_krn** PL/SQL package. The functional name of the schema to be exported needs to be passed as parameter to the procedure.

So the export job is executed as follows:

```
BEGIN
    dpp_job_krn.export_schema('func_name');
END;
/
```

or

```
EXEC dpp_job_krn.export_schema('func_name');
```

Where **func_name** must be replaced with the functional name of the schema to be exported.

Once the export job is started, a new line is introduced in the **dpp_job_runs** log table. The **status** field indicates the status of the job. Those are the possible values:

Status	Description
BSY	busy, the job is currently running
OK	the job terminated successfully
ERR	the job terminated with errors

The **dpp_job_logs** table contains the execution log which is eventually also sent by mail.

5.2. Importing dump file

A common use case of the DPP Utility is the import of a Data Pump dump file from another system.

5.2.1. Dump file storage

The Data Pump dump file generated by the DPP Utility is stored in an Oracle directory the current user has read and write access to. Oracle stores the dump-files on the server-side, this means that the UNIX user running Oracle must have read/write privileges on said directory.

This directory may have different characteristics depending on the current environment:

- **DC - Data Center database:** The test and production environments may have different storage locations.
- **AWS - Amazon Web Services database:** The export directory is the one configured by Amazon, usually **DATA_PUMP_DIR**.
- **COP - Cloud On Premise database:** According to the naming conventions in use at the European Commission expressed in the Oracle COP user guide, the export directory is usually **DP_DIR_S3**.

Still depending on the environment, this is the matching between the Oracle directory and the OS/Unix one:

Environment	Oracle directory	OS/Unix directory
Data Center	DBCC_DBIN	/ec/dev/server/oracle/digit-files/dbccd/dbin
Data Center	DBCC_DBOUT	/ec/dev/server/oracle/digit-files/dbccd/dbout
Awazon Web Services	DATA_PUMP_DIR	/rdsdbdata/datapump
Cloud On Premise	DP_DIR_S3	/oracle/dp

For a COP database, as described in the Oracle COP user guide, the Oracle directory is mapped on a S3 bucket mounted on the */oracle/dp* OS directory. The user guide explains how such an S3 bucket should be requested and mapped in the COP Oracle database.

The dump file follows some naming conventions. The dump file name is made of several concatenated parts:

- the name of the exported database schema
- the export date in **YYYYMMDD** format
- a three digits number indicating the number of the export on a same day
- a three digits number indicating the parallelism number; when several CPUs are used to generate the dump file, each of them produces a file identified with this parallelism number
- the **.exp** extension; during the export operation, the file has temporarily the **.bsy** extension

For example, the **APP_DBECOE_D20240517002002.exp** dump file name may be interpreted as follows:

- The exported schema is **APP_DBECOE_D**.
- The export date is the 17th of May 2024.
- This the third export of the day, the numbering starting at **000**.
- This is the second dump file of this export made with parallelism.

5.2.2. Setting up import

An DPP Utility import job is defined with several parameters that need to be inserted in several database tables.

5.2.2.1. DPP_INSTANCES - Database instance

The database instance must be defined in the **dpp_instances** table as follows:

Field	Description	Example value
ITE_NAME	database instance name	DBCC_DIGIT_01_D
DESCR_ENG	description en English	DBE CoE's development database
DESCR_FRA	description in French	base de données de développement du DBE CoE
PRODUCTION_FLAG	flag indicating whether this is a production instance (**Y** or **N**)	N
ENV_NAME	environment name (**DC**, **AWS** or **COP**)	DC
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

At least one instance must be defined. If the source and target schemas are located in different database instances, two instances must be defined in this table.

5.2.2.2. DPP_PARAMETERS - General parameters

Several general parameters must be defined in the **dpp_parameters** table as follows:

Field	Description	Example value
PRR_NAME	parameter name	g_dpp_in_dir
PRR_VALUE	parameter value	DATA_PUMP_DIR
DESCR_ENG	description in English	Oracle directory for export
DESCR_FRA	description in French	répertoire Oracle d'export
ITE_NAME	The name of the database instance defined in **dpp_instance** if the parameter relates to a particular instance. If the parameter is general, this field may be left empty.	DBCC_DIGIT_01_D
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

The following parameters need to be defined for an import job:

Parameter	Mandatory	Comment
g_dpp_in_dir	yes	name of the Oracle directory the dump files must be loaded from
smtp.dev_recipient	no	The DPP Utility can generate a mail at the end of the job indicating whether it was executed successfully or with error. If the job is not executed in production environment, the mail is not sent to the regular recipients to avoid spamming. If a mail address is mentioned for the parameter, the mail will be sent to this address instead of the regular recipients.
smtp.host	no	The SMTP server that needs to be used to send the mail at the end of the job. If it is not mentioned, the default (**localhost**) server is used.
smtp.port	No	This optional parameter stores the IP port of the SMTP server.
smtp.domain	No	This optional parameter stores the SMTP server domain.
smtp.username	No	If the SMTP server requires some authentication, the user ID is stored in this parameter.
smtp.wallet_path	No	Some SMTP servers need a certificate to complete the authentication. This parameter stores the path of the Oracle directory the **cwallet.sso** certificate file is stored in.
smtp.sender	No	This parameter stores the email address of the mail sender.
smtp.default_recipient	No	This is the default recipient a mail is sent to when the list of recipients cannot be determined. This is the case when the target schema functional name given as parameter does not exist.

5.2.2.3. DPP_ROLES - Database role

If a specific database role must be used to execute the import job, it must be defined in the **dpp_roles** table as follows:

Field	Description	Example value
RLE_NAME	role name	IMPORT_ROLE
DESCR_ENG	description in English	import role
DESCR_FRA	description in French	role d'import
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

Defining a role is optional since they are only needed when some code must be executed via the **dpp_actions** configuration before or after an export or import operation. Such an action could be restoring some privileges to the role.

5.2.2.4. DPP_SCHEMAS - Schemas

The source database schema and the target schema need to be defined in the **dpp_schemas** table as follows:

Field	Description	Example value
SMA_ID	schema unique identifier	1
ITE_NAME	instance name as defined in the **dpp_instances** table	DBCC_DIGIT_01_D
RLE_NAME	optional role name as defined in the **dpp_roles** table	IMPORT_ROLE
STE_NAME	schema type as defined in the **dpp_schema_types** table	MAIN
FUNCTIONAL_NAME	an unique functional name given to the schema	SRCDEV
SMA_NAME	real name of the database schema	APP_DPP4TGT_D
PRODUCTION_FLAG	flag indicating whether the schema is a production one (**Y** or **N**)	N
DATE_FROM	start date of the schema validity	2024-05-22 13:07:34
DATE_TO	optional end date of the schema validity, the schema has no end date if omitted	2030-12-31 23:59:59
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

At least two schemas must be defined in this table, the source and the target schema.

5.2.2.5. DPP_ACTIONS - Specific actions

Some actions must be executed before or after the import operation. They need to be defined in the **dpp_actions** table as follows:

Field	Description	Example value
SMA_ID	the schema ID as defined in the **dpp_schemas** table	1
ATN_USAGE	the flow direction (**I** for an import, **E** for an export, **T** for a transfer and **M** for a monitoring)	I
ATN_TYPE	prefix or postfix action (**PREFIX** or **POSTFIX**)	POSTFIX
EXECUTION_ORDER	execution order	1
BLOCK_TEXT	the code to be executed	BEGIN post_import(); END;
ACTIVE_FLAG	whether the action is active (**Y** or **N**)	Y
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

For each schema, several actions or none may be defined.

If the action code stored in **BLOCK_TEXT** contains some sensitive pieces of information, like passwords for example, they can be encrypted using the SEC Utility. The sensitive information can be replaced with **{DPPCRYPT:var}** where **var** needs to be replaced with the username or variable associated with the sensitive value. The value will be decrypted when the action will be executed.

5.2.2.6. DPP_RECIPIENTS - Mail recipients

At the end of the import process, the DPP Utility can send a mail to some recipients indicating whether the job was executed successfully or with some errors. The mail recipients are defined in the **dpp_recipients** table as follows:

Field	Description	Example value
SMA_ID	the schema ID as defined in the **dpp_schemas** table	1
EMAIL_ADDR	the email address	fname.lname@ec.europa.eu
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

No, one or several recipients may be defined for each schema. If none is defined, no mails will be sent at the end of the import process.

5.2.2.7. DPP_SCHEMA_OPTIONS - Schema options

Several options can be defined to fine tune the import job. Those options are more or less the ones of the regular Oracle Data Pump tool with some additional ones. Those options are defined in the **dpp_schema_options** table as follows:

Field	Description	Example value
SMA_ID	the schema ID as defined in the **dpp_schemas** table	1
OTN_NAME	option name	EMAIL_RESULT
STN_VALUE	option value	Y
STN_USAGE	the flow direction (**I** for an import, **E** for an export, **T** for a transfer)	I
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

Several options can be defined for a schema and a flow. They are all defined in the **dpp_options** table and the possible values, when enumerated, are available in the **dpp_option_allowed_values** table. Those are the options that are available for an export job:

Option name	Description	Possible values
BLOCK	This option blocks any import or export operation. When scheduling background jobs it is not needed to kill the job to stop a export/import action. Just setting the option will skip any export/import.	**Y** for yes or **N** for no
CONSTRAINTS	Drop all REFERENTIAL integrity constraints in the user's schema BEFORE the import of objects via data pump.	**DROP** (Default: undefined)
EMAIL_RESULT	Emails the final result to the designated email list. If the job ended in **ERROR** the log from **dpp_job_logs** will be appended to the mail.	**Y** for yes or **N** for no
EXEC_POSTFIX	After the data pump-job have imported database objects into the target schema. It is possible to execute custom SQL scripts.	**Y** for yes or **N** for no
EXEC_POSTFIX_START	Applies only within a defined scope of **EXEC_POSTFIX**. If **EXEC_POSTFIX** is not defined this parameter is ignored. This option provides with the possibility to skip custom postfix SQL scripts that have a lower ordinal sequence number than the one specified with this option.	a valid ordinal script sequence number (Default: undefined)
EXEC_PREFIX	The export job has now the possibility to execute scripts in the target schema, before exporting data.	**Y** for yes or **N** for no
JOBS	Optionally stops all scheduled jobs defined in the target schema, this happens before an import AND right after a data pump import. (See sequence steps).	**DROP** (Default: undefined)
LOCK_SCHEMA	Locks the target schema and kills all connected sessions.	**YES** (Default: undefined)
METADATA_FILTER	Filter to exclude tables whose name is compliant with the given pattern.	NOT LIKE "xxx%" pattern. Example: NOT LIKE "TMP%"
METALINK_429846_1_CORRECTION	This option is a workaround to statistics, some versions of 10g freeze while datapump tries to import the element EXPORT/TABLE/STATISTICS/TABLE_STATISTICS.	**Y** for yes or **N** for no
MV	Drops all materialized views in the user's schema BEFORE the import of objects via data pump.	**DROP** (Default: undefined)
NETWORK_LINK	Name of a database link to be used for direct export/import.	A database link name, including the domain. Example: **SRC.CC.CEC.EU.INT**
PL_SQL_SOURCE	Drops all local packages, functions, procedures and triggers in the target schema before the import of objects via data pump.	**DROP** (Default: undefined)
PRIVATE_DB_LINKS	Drops all private database links in the target schema BEFORE the import of objects via data pump.	**DROP** (Default: undefined)
RECOMPILE_PL_SQL	After a data pump import, optionally recompiles possible invalid packages/procedures/functions/triggers.	**Y** for yes or **N** for no
RECYCLEBIN	Purges the ORACLE RECYCLEBIN after dropping of database objects (specified by options or prefix SQL scripts).	**PURGE** (Default: undefined)
REMAP_TABLESPACE	Remaps an object exported in tablespace X to tablespace Y during import.	(Default: undefined). A list of tablespace mappings in the form `Oldtablespace1=>Newtab lespace1, Oldtablespace2=>NewTab lespace2, .., OldtablespaceN=>NewTablespaceN` (Example: `SP2TAB=>SP3TAB, SP2IDX=>SP3IDX`).
SEGMENT_ATTRIBUTES	Ignores the tablespace and Oracle storage clause of tables and indexes when importing via data pump. This is for importing into tablespace NOT under automatic segment space management, ASSM. This mitigates the **ORA-01658: unable to create INITIAL extent for segment in …… error**.	**IGNORE** (Default: undefined)
SEQUENCES	Drops all sequences in the target schema before the import of objects via data pump.	**DROP** (Default: undefined)
SIMULATION_IMPORT	Simulates the execution of the import job (including specified options, like dropping objects before	**Y** for yes or **N** for no

Option name	Description	Possible values
	import etc.) without actually importing data via data pump.	
STORAGE	Ignores the Oracle storage clause of tables and indexes when importing via data pump.	**IGNORE** (Default: undefined)
SYNONYMS	Drops all synonyms in the users schema before the import of objects via data pump.	**DROP** (Default: undefined)
TABLES	Drops all tables in the user's schema BEFORE the import of objects via data pump.	**DROP** (Default: undefined)
TRIGGERS	Drops all database triggers in the user's schema BEFORE the import of objects via data pump.	**DROP** (Default: undefined)
TYPES	Drops all types in the user's schema before the import of objects via data pump.	**DROP** (Default: undefined)
VIEWS	Drops all views in the target schema before the import of objects via data pump.	**DROP** (Default: undefined)
TIMEOUT_MONITORING	Activates or deactivates the timeout monitoring for the current schema.	**Y** for yes or **N** for no
TIMEOUT_DELAY	Number of minutes defining the delay after which the job is considered being in timeout.	Numeric value, number of minutes.
DATA_FILTER	Create a filter on a table in order to exclude some data. Such an entry must be created for each table that must be filtered. The Oracle documentation should be checked for more details.	The value is a concatenation of three values separated with the "#" character. These three values are the name of the table to be filtered (the filter is for all tables if omitted), the filter type (**INCLUDE_ROWS**, **PARTITION_EXPR**, **PARTITION_LIST**, **SAMPLE** or **SUBQUERY**) and the filter value depending on the filter type. If the table name is not mentioned, the separator must be present anyway. Example: TST_EMPLOYEE#SUBQUERY#WHERE EMP_ID < 400
ENCRYPTION_PASSWORD	This is the encryption password to be used to import a dump that has been protected at [export time] (#51-exporting-dump-file with the password encryption method.	the same password as the one used for the export
LOGTIME	Define the type of timestamps to be mentioned in the logs generated by the Data Pump as described in the Oracle documentation.	**NONE**, **STATUS**, **LOGFILE** or **ALL**
DATA_REMAP	With this option, a transformation can be applied to a column at import time as explained in the Oracle documentation. Such an entry must be created for each column that must be transformed.	The value is the concatenation of four fields separated with the "#" character. Those fields are the name of the remap (**COLUMN_FUNCTION** usually), the name of the table the column belongs to, the column name and the name of a PL/SQL function in a package that applies the transformation. The function must have a single parameter of the same type as the column to be transformed and must return a value of the same type. For example: COLUMN_FUNCTION#TST_EMPLOYEE#FIRST_NAME#DATA_REMAP_TEST.TO_UPPER

5.2.2.8. DPP_SCHEMA_RELATIONS - Relationships between schemas

It is possible to define relationships between database schemas, such as a link between a main schema and gateway/child schemas. Those relationships are defined in the **dpp_schema_relations** table as follows:

Field	Description	Example value
SMA_ID_FROM	The ID of the source schema of the relationship as defined in the **dpp_schemas** table.	1
SMA_ID_TO	The ID of the target schema of the relationship as defined in the **dpp_schemas** table.	2
DATE_FROM	Start date of the relationship validity period.	2024-05-22 08:38:50
DATE_TO	End date of the relationship validity period. If this date is omitted, there is no end of validity.	2030-05-01 23:59:59
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

Defining relationships between schemas is optional and one or several relationships can be defined for each schema. When there are some gateway schemas in the database having access to a schema that can be the target of an import operation, it is essential to define these relationships in order to allow kill and lock actions before starting the import process.

5.2.2.9. DPP_NODROP_OBJECTS - Protected objects

In the **dpp_schema_options** table, some options tells the DPP Utility that some objects in the target schema must be dropped before importing the source schema or dump file. Those objects can be tables, constraints, jobs, materialized views, PL/SQL objects like packages, functions or procedures, sequences, synonyms, triggers, types or views.

However, it is possible to prevent some specific objects for being dropped. This can be done by inserting the name of those objects in the **dpp_nodrop_objects** table as follows:

Field	Description	Example value
SMA_ID	The ID of the target schema of the relationship as defined in the **dpp_schemas** table.	2
OBJECT_NAME	The name of the object that may not be dropped.	PMG_TB_DATA_MODEL
OBJECT_TYPE	The type of the object that may not be dropped.	TABLE
ACTIVE_FLAG	Whether this rule is active (**Y** for yes and **N** for no).	Y
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

5.2.3. Preparing schema for import

Some actions may be required to prepare a schema for the import of a dump.

If there some queues in use in the schema to be imported, the following code must be executed while logged with the user of the schema the dump must be imported in:

```
BEGIN
   -- needed only if there are Oracle Advanced Queues to manage
   DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('ENQUEUE_ANY', '&&APP_DPP',FALSE);
   DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('DEQUEUE_ANY', '&&APP_DPP',FALSE);
   DBMS_AQADM.GRANT_SYSTEM_PRIVILEGE('MANAGE_ANY', '&&APP_DPP',FALSE);
END;
/
```

5.2.4. Running the import

Once all needed parameters have been defined, the import job can be executed. It is implemented by the **import_schema** procedure of the **dpp_job_krn** PL/SQL package. The functional name of the source schema and the target schema need to be passed as parameters to the procedure.

So the import job is executed as follows:

```
BEGIN
    dpp_job_krn.import_schema('src_func_name', 'trg_func_name');
END;
/
```

or

```
EXEC dpp_job_krn.import_schema('src_func_name', 'trg_func_name');
```

Where **src_func_name** must be replaced with the functional name of the source schema and **trg_func_name** must be replaced with the functional name of the target schema.

Once the import job is started, a new line is introduced in the **dpp_job_runs** log table. The **status** field indicates the status of the job. Those are the possible values:

Status	Description
BSY	busy, the job is currently running
OK	the job terminated successfully
ERR	the job terminated with errors

The **dpp_job_logs** table contains the execution log which is eventually also sent by mail.

5.2.5. Import through a network link

The schema import can also be performed without the use of a dump file. In this case, the import is performed directly from the source database instance schema to the target database schema.

In order to perform the import through the network link, the following configurations are required:

1. Creating a database link in the target database schema referencing the source database instance schema.
2. In the **NETWORK_LINK** option in the **DPP_SCHEMA_OPTIONS** table, the name of the database link created above must be mentioned for the import flow of the target schema.

5.3. Transferring dump file

When a dump file has been exported and needs to be imported in another database instance, if first needs to be transferred from the source database instance to the target database instance. In a Data Center installation, the dump file needs also to be transferred from the export directory to the import directory as described in the **dpp_parameters** database table.

Transferring the dump file is another common use case of the DPP Utility.

5.3.1. Setting up transfer

An DPP Utility job is defined with several parameters that need to be inserted in several database tables.

5.3.1.1. DPP_INSTANCES - Database instance

The database instance must be defined in the **dpp_instances** table as follows:

Field	Description	Example value
ITE_NAME	database instance name	DBCC_DIGIT_01_D
DESCR_ENG	description en English	DBE CoE's development database
DESCR_FRA	description in French	base de données de développement du DBE CoE
PRODUCTION_FLAG	flag indicating whether this is a production instance (**Y** or **N**)	N
ENV_NAME	environment name (**DC**, **AWS** or **COP**)	DC
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

At least one instance must be defined for the source schema.

5.3.1.2. DPP_PARAMETERS - General parameters

Several general parameters must be defined in the **dpp_parameters** table as follows:

Field	Description	Example value
PRR_NAME	parameter name	g_dpp_out_dir
PRR_VALUE	parameter value	DATA_PUMP_DIR
DESCR_ENG	description in English	Oracle directory for export
DESCR_FRA	description in French	répertoire Oracle d'export
ITE_NAME	The name of the database instance defined in **dpp_instance** if the parameter relates to a particular instance. If the parameter is general, this field may be left empty.	DBCC_DIGIT_01_D
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27

Field	Description	Example value
USER_MODIF	last modification user ID	malmjea

The following parameters need to be defined for an transfer job:

Parameter	Mandatory	Comment
g_dpp_our_dir	yes	name of the Oracle directory the dump files must be transferred from
smtp.dev_recipient	no	The DPP Utility can generate a mail at the end of the job indicating whether it was executed successfully or with error. If the job is not executed in production environment, the mail is not sent to the regular recipients to avoid spamming. If a mail address is mentioned for the parameter, the mail will be sent to this address instead of the regular recipients.
smtp.host	no	The SMTP server that needs to be used to send the mail at the end of the job. If it is not mentioned, the default (**localhost**) server is used.
smtp.port	No	This optional parameter stores the IP port of the SMTP server.
smtp.domain	No	This optional parameter stores the SMTP server domain.
smtp.username	No	If the SMTP server requires some authentication, the user ID is stored in this parameter.
smtp.wallet_path	No	Some SMTP servers need a certificate to complete the authentication. This parameter stores the path of the Oracle directory the **cwallet.sso** certificate file is stored in.
smtp.sender	No	This parameter stores the email address of the mail sender.
smtp.default_recipient	No	This is the default recipient a mail is sent to when the list of recipients cannot be determined. This is the case when the schema name given as parameter does not exist.

5.3.1.3. DPP_ROLES - Database role

If a specific database role must be used to execute the transfer job, it must be defined in the **dpp_roles** table as follows:

Field	Description	Example value
RLE_NAME	role name	IMPORT_ROLE
DESCR_ENG	description in English	transfer role
DESCR_FRA	description in French	role de transfert
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

Defining a role is optional since they are only needed when some code must be executed via the **dpp_actions** configuration before or after an export or import operation. Such an action could be restoring some privileges to the role.

5.3.1.4. DPP_SCHEMAS - Schemas

The source database schema needs to be defined in the **dpp_schemas** table as follows:

Field	Description	Example value
SMA_ID	schema unique identifier	1
ITE_NAME	instance name as defined in the **dpp_instances** table	DBCC_DIGIT_01_D
RLE_NAME	optional role name as defined in the **dpp_roles** table	TRANSER_ROLE
STE_NAME	schema type as defined in the **dpp_schema_types** table	MAIN
FUNCTIONAL_NAME	an unique functional name given to the schema	SRCDEV
SMA_NAME	real name of the database schema	APP_DPP4TGT_D
PRODUCTION_FLAG	flag indicating whether the schema is a production one (**Y** or **N**)	N
DATE_FROM	start date of the schema validity	2024-05-22 13:07:34
DATE_TO	optional end date of the schema validity, the schema has no end date if omitted	2030-12-31 23:59:59
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

At least one schema must be defined in this table, the source one.

5.3.1.5. DPP_RECIPIENTS - Mail recipients

At the end of the transfer process, the DPP Utility can send a mail to some recipients indicating whether the job was executed successfully or with some errors. The mail recipients are defined in the **dpp_recipients** table as follows:

Field	Description	Example value
SMA_ID	the schema ID as defined in the **dpp_schemas** table	1
EMAIL_ADDR	the email address	fname.lname@ec.europa.eu
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

No, one or several recipients may be defined for each schema. If none is defined, no mails will be sent at the end of the transfer process.

5.3.1.6. DPP_SCHEMA_OPTIONS - Schema options

Several options can be defined to fine tune the transfer job. Those options are defined in the **dpp_schema_options** table as follows:

Field	Description	Example value
SMA_ID	the schema ID as defined in the **dpp_schemas** table	1
OTN_NAME	option name	EMAIL_RESULT
STN_VALUE	option value	Y
STN_USAGE	the flow direction (**I** for an import, **E** for an export, **T** for a transfer)	I
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

Several options can be defined for a schema and a flow. They are all defined in the **dpp_options** table and the possible values, when enumerated, are available in the **dpp_option_allowed_values** table. Those are the options that are available for a transfer job:

Option name	Description	Possible values
EMAIL_RESULT	Emails the final result to the designated email list. If the job ended in **ERROR** the log from **dpp_job_logs** will be appended to the mail.	**Y** for yes or **N** for no
TIMEOUT_MONITORING	Activates or deactivates the timeout monitoring for the current schema.	**Y** for yes or **N** for no
TIMEOUT_DELAY	Number of minutes defining the delay after which the job is considered being in timeout.	Numeric value, number of minutes.

5.3.2. Running the transfer

Once all needed parameters have been defined, the transfer job can be executed. It is implemented by the **transfer_dumpfiles** procedure of the **dpp_job_krn** PL/SQL package. The functional name of the source schema and the name of the database link used to transfer the file need to be passed as parameter to the procedure.

So the import job is executed as follows:

```
BEGIN
    dpp_job_krn.transfer_dumpfiles('func_name', 'db_link_name');
END;
/
```

or

```
EXEC dpp_job_krn.transfer_dumpfiles('func_name', 'db_link_name');
```

Where **func_name** must be replaced with the functional name of the source schema and **db_link_name** must be replaced with the name of the database link used to transfer the dump file from the source database to the target one. It is assumed that the Oracle directories have the same name in both source and target environments.

Once the import job is started, a new line is introduced in the **dpp_job_runs** log table. The **status** field indicates the status of the job. Those are the possible values:

Status	Description
BSY	busy, the job is currently running
OK	the job terminated successfully
ERR	the job terminated with errors

The **dpp_job_logs** table contains the execution log which is eventually also sent by mail.

5.4. Timeout monitoring

A job is part of the DPP Utility and is intended to detect when an export, import or transfer job is in timeout. The monitoring runs every five minutes and detects the jobs that are still in busy status for more than an amount of time. In this case, the corresponding job status is set to error and an alert email is sent.

5.4.1. Setting up timeout monitoring

The timeout monitoring is activated per import, export or transfer job be defining some parameters in the tables described below.

5.4.1.1. DPP_SCHEMA_OPTIONS - Schema options

Several options can be defined in the **dpp_schema_options** table to activate the monitoring as follows:

Field	Description	Example value
SMA_ID	the schema ID as defined in the **dpp_schemas** table	1
OTN_NAME	option name	PARALLEL
STN_VALUE	option value	10
STN_USAGE	the flow direction (**I** for an import, **E** for an export, **T** for a transfer)	E
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

The options related to the timeout monitoring are:

Option name	Description	Possible values
TIMEOUT_MONITORING	This flag indicates whether the timeout monitoring must be activated for the current schema and for the import or export flow.	The possible values are **Y** or **N**
TIMEOUT_DELAY	This option defines a number of minutes which is the timeout delay. When a job is busy for more than this number of minutes, it is considered as being in timeout. This option has no effect when the **TIMEOUT_MONITORING** option is not set to **Y**. The value is a number of minutes. If the **TIMEOUT_MONITORING** option is set to **Y** and the **TIMEOUT_DELAY** option is not defined, a default delay of 300 minutes is used.	number of minutes

5.4.1.2. DPP_RECIPIENTS - Mail recipients

For each schema, a list of mail recipients is defined in the **dpp_recipients**. Those are the persons the timeout alert mail will be sent to. At least one recipient is needed so that the mail is sent. Those mail addresses are defined as follows:

Field	Description	Example value
SMA_ID	the schema ID as defined in the **dpp_schemas** table	1
EMAIL_ADDR	the email address	fname.lname@ec.europa.eu
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

5.4.1.3. DPP_PARAMETERS - General parameters

Several parameters need to be defined in the **dpp_parameters** table in order to activate the timeout monitoring as follows:

Field	Description	Example value
PRR_NAME	parameter name	g_dpp_in_dir
PRR_VALUE	parameter value	DATA_PUMP_DIR
DESCR_ENG	description in English	Oracle directory for export
DESCR_FRA	description in French	répertoire Oracle d'export
ITE_NAME	The name of the database instance defined in **dpp_instance** if the parameter relates to a particular instance. If the parameter is general, this field may be left empty.	DBCC_DIGIT_01_D
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

The following parameters need to be defined for the timeout monitoring:

Parameter	Mandatory	Comment
smtp.host	No	This optional parameter stores the SMTP server host name or, eventually, its IP address. If omitted, the default SMTP server will be used.
smtp.port	No	This optional parameter stores the IP port of the SMTP server.
smtp.domain	No	This optional parameter stores the SMTP server domain.
smtp.username	No	If the SMTP server requires some authentication, the user ID is stored in this parameter.
smtp.wallet_path	No	Some SMTP servers need a certificate to complete the authentication. This parameter stores the path of the Oracle directory the **cwallet.sso** certificate file is stored in.
smtp.sender	No	This parameter stores the email address of the mail sender.
smp.dev_recipient	No	The recipients defined in the **dpp_recipients** table are used in production environment. In other environments, a default recipient is used which is stored in this parameter.

5.4.1.4. DPP_ACTIONS - Specific actions

Some actions can be executed after the detection of a job being in timeout. They need to be defined in the **dpp_actions** table as follows:

Field	Description	Example value
SMA_ID	the schema ID as defined in the **dpp_schemas** table	1
ATN_USAGE	the flow direction (**M** for monitoring)	M
ATN_TYPE	postfix action (**POSTFIX**)	POSTFIX
EXECUTION_ORDER	execution order	1
BLOCK_TEXT	the code to be executed	BEGIN post_monitoring(); END;
ACTIVE_FLAG	whether the action is active (**Y** or **N**)	Y
DATE_CREAT	row creation date	2024-05-21 17:38:48
USER_CREAT	row creation user ID	malmjea
DATE_MODIF	last modification date	2024-05-21 17:39:27
USER_MODIF	last modification user ID	malmjea

For each schema, several actions or none may be defined. The behavior of the post monitoring actions differs from the ones of the export job and the import job. The action concerns the timeout detection of a job in a specific schema which is mentioned in the **sma_id** field. However, the corresponding action will be executed with the database schema user where the DPP Utility is deployed. Furthermore, the usage is always **M** for a post-monitoring action and it is always a **postfix** action.

If the action code stored in **BLOCK_TEXT** contains some sensitive pieces of information, like passwords for example, they can be encrypted using the SEC Utility. The sensitive information can be replaced with **{DPPCRYPT:var}** where **var** needs to be replaced with the username or variable associated with the sensitive value. The value will be decrypted when the action will be executed.

5.5. Secondary operations

5.5.1. Removing dump file

An existing dump file can be removed from the Oracle directory as follows:

```
BEGIN
    dpp_job_krn.remove_dmp_file('file_name', 'directory_name');
END;
/
```

or:

```
EXEC dpp_job_krn.remove_dmp_file('file_name', 'directory_name');
```

Where:

- **file_name** must be replaced with the name of the dump file to be removed.
- **directory_name** must be replaced with the name of the Oracle directory the file is stored in.

5.5.2. Removing old dump files

Old dump files can be removed as follows:

```
BEGIN
    dpp_job_krn.remove_files_older('schema_name', 'date');
END;
/
```

or:

```
EXEC dpp_job_krn.remove_files_older('schema_name', 'date');
```

Where:

- **schema_name** must be replaced with the technical name of the schema whose dump files must be removed.
- **date** must be replaced with a date. All dump files of the schema mentioned in the first parameter made on this date and before will be removed.

Another way to delete old dump files:

```
BEGIN
    dpp_job_krn.remove_files_old_functional('functional_name', 'date');
END;
/
```

or:

```
EXEC dpp_job_krn.remove_files_old_functional('functional_name', 'date');
```

Where:

- **functional_name** must be replaced with the functional name of the schema whose dump files must be removed.
- **date** must be replaced with a date. All dump files of the schema mentioned in the first parameter made on this date and before will be removed.

5.6. Sending mail

Depending on the configuration of the **dpp_schema_options**, **dpp_parameters** and **dpp_recipients**, a mail can be automatically sent by the DPP Utility at the end of an export, import or transfer job. The following chapters describe those configurations:

- for an export job:
  - **dpp_parameters**
  - **dpp_recipients**
  - **dpp_schema_options**
- for an import job:
  - **dpp_parameters**
  - **dpp_recipients**
  - **dpp_schema_options**
- for a transfer job:
  - **dpp_parameters**
  - **dpp_recipients**
  - **dpp_schema_options**

The mail is actually sent by the MAIL Utility.

When the jobs are executed in a non-production environment, the MAIL Utility does not send the mail to the regular recipients. In the **dpp_parameters** table, a default email recipient for non-production environments can be defined in the **smtp.dev_recipient** parameter.

By default, the mail will be sent to this address. To inform the MAIL Utility that the environment is the production one and the mails should be sent to the regular recipients, the following piece of code can be executed before running the DPP Utility:

```
BEGIN
   mail_utility_krn.set_is_prod('Y');
END;
/
```

or:

```
EXEC mail_utility_krn.set_is_prod('Y');
```

5.7. Managing configurations

Each DPP Utility operation is defined by some configuration data as explained in the previous chapters. This configuration data can be inserted, modified or deleted directly in the corresponding database tables. The configuration data can also be managed using a dedicated API which is implemented by a dedicated PL/SQL package called **dpp_cnf_krn**.

5.7.1. Managing database links

Database links are used to perform data pump imports through a network link or to transfer dump files from a database instance to another one. The database links are created in the target schemas managed by the DPP Utility.

The database links can be created or deleted in the target schema using two dedicated procedures:

- **create_database_link** procedure
- **drop_database_link** procedure

5.7.2. Managing atomic configuration items

The configuration API proposes several procedures or functions to create, update and delete atomic items in the following configuration tables:

- Instances: insert_instance, update_instance and delete_instance
- Schema types: insert_schema_type, update_schema_type and delete_schema_type

- Roles: insert_role, update_role and delete_role
- Schemas: insert_schema, update_schema and delete_schema
- "no drop" objects: insert_nodrop_object, update_nodrop_object and delete_nodrop_object
- Actions: insert_action, update_action, insert_http_request_action, insert_http_request_action_enc, insert_http_gitlab_req_action, insert_http_gitlab_req_acton_enc, update_action_exec_order and delete_action
- Parameters: insert_parameter, update_parameter and delete_parameter
- Email recipients: insert_recipient and delete_recipient
- Schema options: insert_schema_option, update_schema_option and delete_schema_option
- Schema relations: insert_schema_relation, update_schema_relation and delete_schema_relation

5.7.3. Duplicating atomic configuration items

Most of the configuration items are linked to a schema. Some procedures and functions of the API allow the duplication of those atomic configuration items and to link them to another schema:

- duplicate_schema: This function creates a copy of an existing schema in the **dpp_schemas** table. The newly created schema gets a new ID and functional name. The functional name must be passed as parameter but the ID can be automatically generated.
- duplicate_action: This procedure creates a copy of an existing action in the **dpp_actions** table. The new action is linked to another schema passed as parameter.
- duplicate_nodrop_object: This procedure creates a copy of an existing "no drop" object in the **dpp_nodrop_objects** table. The new "no drop" object is linked to another schema passed as parameter.
- duplicate_recipient: This procedure creates a copy of an existing email recipient in the **dpp_recipients** table. The new recipient is linked to another schema passed as parameter.
- duplicate_schema_option: This procedure creates a copy of an existing schema option in the **dpp_schema_options** table. The new option is linked to another schema passed as parameter.

In those functions, the source and target schemas can be identified by their ID or their functional name which is unique. For the duplication of a schema, the new functional name must be passed as parameter since it cannot be determined by the function.

5.7.4. Duplicating a full configuration

The configuration API allows the duplication of a schema and all attached configuration items in one single operation by calling the **duplicate_schema_config** function. This function executes the atomic configuration items duplication procedures for each item linked to the schema to be duplicated.

This allows the creation of some kind of configuration templates that can be linked to a virtual schema and that can be duplicated and linked to a real schema when needed. This is useful when more or less the same configuration is defined for several schemas.

5.7.5. Removing a full configuration

The configuration API allows also the deletion of a schema and all attached configuration items by calling the **delete_schema_config** procedure.

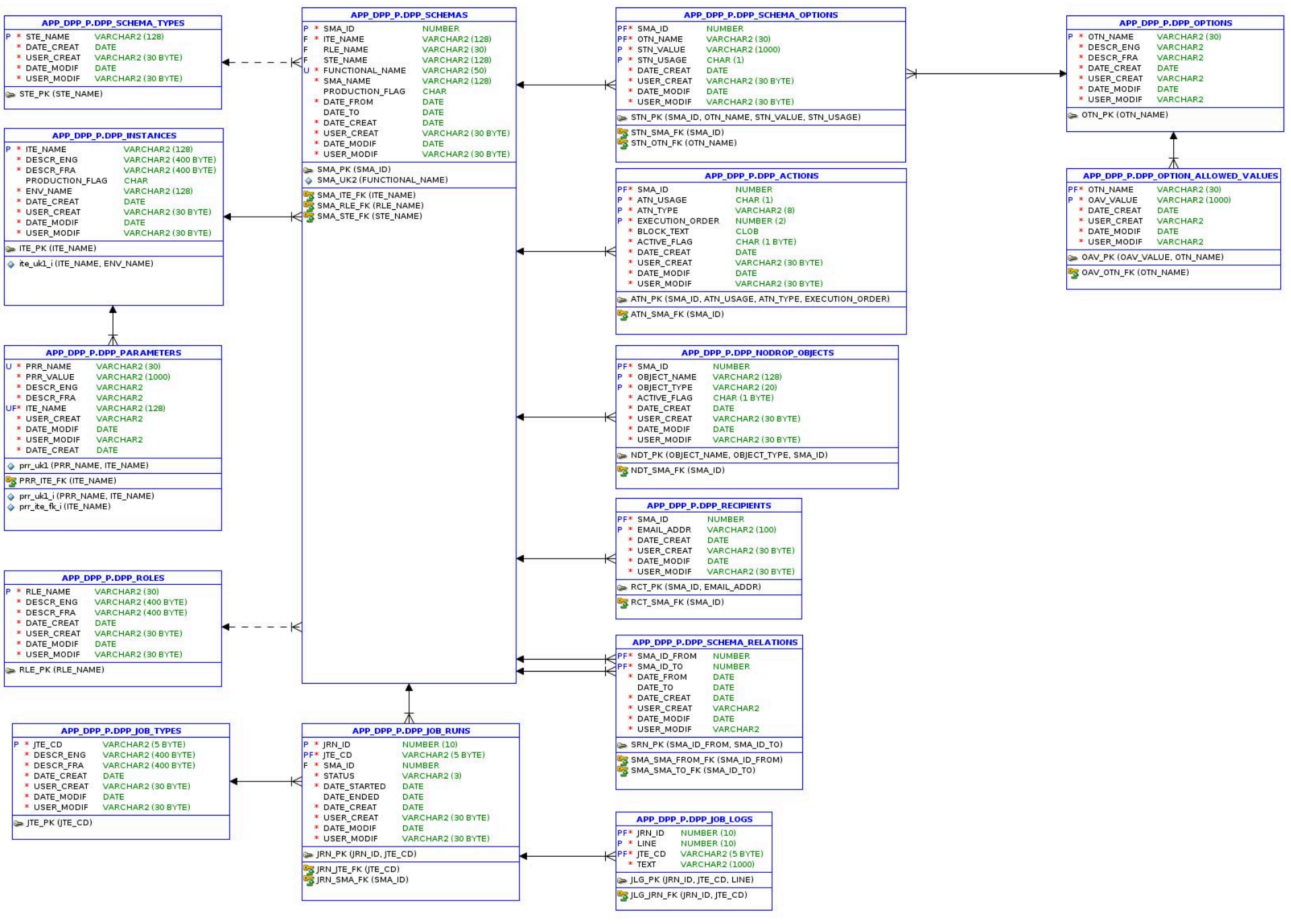
5.7.6. Cleaning up execution logs

The **dpp_job_runs** and **dpp_job_logs** log tables can be cleaned up by executing the **clean_up_job_runs** procedure.

6. Technical implementation

This chapter describes the technical implementation of the DPP Utility.

6.1. Data model



6.1.1. DPP_ACTIONS

This table contains the PL/SQL blocks to be executed before or after an export or an import process.

Column name	Type	Description
SMA_ID	NUMBER NOT NULL	Schema unique identifier
ATN_USAGE	CHAR(1)	Job type: Import / Export / Transfer / Monitoring

Column name	Type	Description
ATN_TYPE	VARCHAR2(8)	prefix or postfix action (PREFIX/POSTFIX)
EXECUTION_ORDER	NUMBER(2)	Order of block execution
BLOCK_TEXT	CLOB	Block code
ACTIVE_FLAG	CHAR(1)	Is the block active flag (Y/N)
DATE_CREAT	DATE NOT NULL	The date this entry was created
USER_CREAT	VARCHAR2(128) NOT NULL	The user who created this entry
DATE_MODIF	DATE NOT NULL	The last time this row was modified by **USER_MODIF** user
USER_MODIF	VARCHAR2(128) NOT NULL	The user who last modified this row

6.1.2. DPP_INSTANCES

This table contains the list of database instances for which the tool is configured.

Column name	Type	Description
ITE_NAME	VARCHAR2(128) NOT NULL	Database instance name
DESCR_ENG	VARCHAR2(2000) NOT NULL	Description in the English language
DESCR_FRA	VARCHAR2(2000) NOT NULL	Description in the French language
PRODUCTION_FLAG	CHAR(1)	Is this a production instance? (Y/N)
ENV_NAME	VARCHAR2(128) NOT NULL	Environment name (DC, COP, AWS)
DATE_CREAT	DATE NOT NULL	The date this entry was created
USER_CREAT	VARCHAR2(128) NOT NULL	The user who created this entry
DATE_MODIF	DATE NOT NULL	The last time this row was modified by **USER_MODIF** user
USER_MODIF	VARCHAR2(128) NOT NULL	The user who last modified this row

6.1.3. DPP_RECIPIENTS

This table contains a list of email addresses which should receive a notification when an export/import/file transfer process is performed for the related schema.

Column name	Type	Description
SMA_ID	NUMBER NOT NULL	Schema unique identifier
EMAIL_ADDR	VARCHAR2(100) NOT NULL	Email address
DATE_CREAT	DATE NOT NULL	The date this entry was created
USER_CREAT	VARCHAR2(128) NOT NULL	The user who created this entry
DATE_MODIF	DATE NOT NULL	The last time this row was modified by **USER_MODIF** user
USER_MODIF	VARCHAR2(128) NOT NULL	The user who last modified this row

6.1.4. DPP_ROLES

This table contains the list of roles linked to the given schema. This is typically used during the post import phase, to restore grants.

Column name	Type	Description
RLE_NAME	VARCHAR2(30) NOT NULL	Role Name
DESCR_ENG	VARCHAR2(2000) NOT NULL	Description in the English language
DESCR_FRA	VARCHAR2(2000) NOT NULL	Description in the French language
DATE_CREAT	DATE NOT NULL	The date this entry was created
USER_CREAT	VARCHAR2(128) NOT NULL	The user who created this entry
DATE_MODIF	DATE NOT NULL	The last time this row was modified by **USER_MODIF** user
USER_MODIF	VARCHAR2(128) NOT NULL	The user who last modified this row

6.1.5. DPP_SCHEMA_TYPES

This table contains the list of types of schemas. Typically there is the **MAIN** schema, but then you can have others for the application (**WEB**), a specialized module (**PRESENCE** for Sysper), other databases (**COMREF**, **NAP**, ….), lobs objects (**LOB**), …. In Sysper or Folio case, this data is used during the post-import process to restore grants and regenerate synonyms.

Column name	Type	Description
STE_NAME	VARCHAR2(128) NOT NULL	Schema Type Name
DATE_CREAT	DATE NOT NULL	The date this entry was created
USER_CREAT	VARCHAR2(128) NOT NULL	The user who created this entry
DATE_MODIF	DATE NOT NULL	The last time this row was modified by **USER_MODIF** user
USER_MODIF	VARCHAR2(128) NOT NULL	The user who last modified this row

6.1.6. DPP_SCHEMAS

This table maps functional names to real Oracle accounts within a given instance. The password is never stored in this table (or any other table for that matter). The production flag might look redundant but it happens that production instances contain also "helpdesk" or equivalent schemas, in which case it must be possible to make the difference.

Caution:

- Starting from the version 24.0 of the utility, the functional name must be unique.
- Starting from the version 24.2 or the utility, the schema type is optional.

Column name	Type	Description
SMA_ID	NUMBER NOT NULL	Schema unique identifier
SMA_NAME	VARCHAR2(128) NOT NULL	Name of the oracle schema

Column name	Type	Description
FUNCTIONAL_NAME	VARCHAR2(50) NOT NULL	Functional name of the schema
ITE_NAME	VARCHAR2(128) NOT NULL	Instance name as known in view **V$INSTANCE**
RLE_NAME	VARCHAR2(30)	Role linked to the schema
STE_NAME	VARCHAR2(128)	Schema Type
PRODUCTION_FLAG	CHAR(1)	Is this a production schema? (Y/N)
DATE_FROM	DATE NOT NULL	Begin of validity
DATE_TO	DATE	End of validity
DATE_CREAT	DATE NOT NULL	The date this entry was created
USER_CREAT	VARCHAR2(128) NOT NULL	The user who created this entry
DATE_MODIF	DATE NOT NULL	The last time this row was modified by **USER_MODIF** user
USER_MODIF	VARCHAR2(128) NOT NULL	The user who last modified this row

6.1.7. DPP_SCHEMA_RELATIONS

This table allows defining relations between schemas, such as link main schema to related gateway ones….

Column name	Type	Description
SMA_ID_FROM	NUMBER NOT NULL	Schema unique identifier
SMA_ID_TO	VARCHAR2(128) NOT NULL	Linked schema unique identifier
DATE_FROM	DATE NOT NULL	Begin of validity
DATE_TO	DATE	End of validity
DATE_CREAT	DATE NOT NULL	The date this entry was created
USER_CREAT	VARCHAR2(128) NOT NULL	The user who created this entry
DATE_MODIF	DATE NOT NULL	The last time this row was modified by **USER_MODIF** user
USER_MODIF	VARCHAR2(128) NOT NULL	The user who last modified this row

6.1.8. DPP_JOB_TYPES

This table shows the types of data pump jobs possible, there are only 3 types defined currently (import, export and file transfer).

Column name	Type	Description
JTE_CD	VARCHAR2(10) NOT NULL	The type of Job. There are 3 entries: **IMPJB** (import job), **EXPJB** (export job) and **TRFJB** (file transfer job)
DESCR_ENG	VARCHAR2(2000) NOT NULL	Description in the English language
DESCR_FRA	VARCHAR2(2000) NOT NULL	Description in the French language
DATE_CREAT	DATE NOT NULL	The date this entry was created
USER_CREAT	VARCHAR2(128) NOT NULL	The user who created this entry
DATE_MODIF	DATE NOT NULL	The last time this row was modified by **USER_MODIF** user
USER_MODIF	VARCHAR2(128) NOT NULL	The user who last modified this row

6.1.9. DPP_JOB_RUNS

This table contains the actual job runs.

Column name	Type	Description
JRN_ID	NUMBER NOT NULL	First part if the primary key, unique identifier of this batch job.
JTE_CD	VARCHAR2(10) NOT NULL	Second and last part of primary key. References the type of JOB (References **DPP_JOB_TYPES.JTE_CD**)
SMA_ID	NUMBER	Schema unique identifier
DATE_STARTED	DATE NOT NULL	Start time of the job
DATE_ENDED	DATE	End time of the job
STATUS	VARCHAR2(3)	A batch job can 3 statuses: **BSY** (BUSY, Job is currently executing), **OK** (Job executed correctly) and **ERR** (Error, Job stopped due to an error, look at the **DPP_JOB_LOGS** for details of the error)
DATE_CREAT	DATE NOT NULL	The time this entry was created (same as **DATE_STARTED**)
USER_CREAT	VARCHAR2(128) NOT NULL	The user that initiated the job
DATE_MODIF	DATE NOT NULL	When the **STATUS** column is adjusted this field changes!
USER_MODIF	VARCHAR2(128) NOT NULL	The user that modified the value of the **STATUS** column.

The **jrn_id** field is a concatenation of several values:

- Unique identifier of this batch job, the format is **YYYYMMDDNNN**:
  - **YYYYDDMM** (DATE)
  - **SEQUENCE** NUMBER

This means that within a date you can have more than one identifiable export, import or transfer job.

6.1.10. DPP_JOB_LOGS

This table contains the detail information (logging) on the job specified in **DPP_JOB_RUNS**.

Column name	Type	Description
JRN_ID	NUMBER NOT NULL	First part of primary key. References the type of JOB (References **DPP_JOB_RUNS.JRN_ID**)

Column name	Type	Description
JTE_CD	VARCHAR2(10) NOT NULL	Second and part of primary key. References the type of JOB (References **DPP_JOB_RUNS.JTE_CD**)
LINE	NUMBER(10) NOT NULL	Third part of the primary key. Ordinal line number of a log message
TEXT	VARCHAR2(1000)	Detailed log message

6.1.11. DPP_SCHEMA_OPTIONS

This table contains the options chosen to invoke a batch job type on this particular Oracle instance. The options are defined for each logical schema name individually (see **DPP_SCHEMAS**). It is essentially a list of option/value pairs.

Column name	Type	Description
SMA_ID	NUMBER NOT NULL	First part of primary key. This gives the schema for which the default IMPORT/EXPORT/TRANSFER options will apply.
OTN_NAME	VARCHAR2(30) NOT NULL	Second part of primary key. Option chosen.
STN_VALUE	VARCHAR2(1000)	Third part of the key. Value of the option.
STN_USAGE	CHAR(1)	Fourth and last part of the key. Import/export/transfer use type (**I**, **E** or **T**)
DATE_CREAT	DATE NOT NULL	The date this entry was created
USER_CREAT	VARCHAR2(128) NOT NULL	The user who created this entry
DATE_MODIF	DATE NOT NULL	The last time this row was modified by **USER_MODIF** user
USER_MODIF	VARCHAR2(128) NOT NULL	The user who last modified this row

The possible options are stored in the **dpp_options** table and depend on the type of operation that is executed. Those possible values are described in the corresponding chapters:

- **export:** possible values of **dpp_schema_options** for an export operation
- **import:** possible values of **dpp_schema_options** for an import operation
- **transfer:** possible values of **dpp_schema_options** for a transfer operation

6.1.12. DPP_OPTIONS

This table contains all possible options when invoking a batch.

Column name	Type	Description
OTN_NAME	VARCHAR2(30) NOT NULL	Option name
DESCR_ENG	VARCHAR2(2000) NOT NULL	Description in the English language
DESCR_FRA	VARCHAR2(2000) NOT NULL	Description in the French language
DATE_CREAT	DATE NOT NULL	The date this entry was created
USER_CREAT	VARCHAR2(128) NOT NULL	The user who created this entry
DATE_MODIF	DATE NOT NULL	The last time this row was modified by **USER_MODIF** user
USER_MODIF	VARCHAR2(128) NOT NULL	The user who last modified this row

6.1.13. DPP_OPTION_ALLOWED_VALUES

This table contains all possible values for the options, if pertinent.

Column name	Type	Description
OTN_NAME	VARCHAR2(30) NOT NULL	Option name
OAV_VALUE	VARCHAR2 (1000 Byte)	Value of the option
DATE_CREAT	DATE NOT NULL	The date this entry was created
USER_CREAT	VARCHAR2(128) NOT NULL	The user who created this entry
DATE_MODIF	DATE NOT NULL	The last time this row was modified by **USER_MODIF** user
USER_MODIF	VARCHAR2(128) NOT NULL	The user who last modified this row

6.1.14. DPP_NODROP_OBJECTS

This list excludes objects from being dropped if otherwise specified by the (example) TABLE=DROP value.

Column name	Type	Description
SMA_ID	NUMBER NOT NULL	An oracle schema on a given database instance. Part 1/3 of the primary key.
OBJECT_NAME	VARCHAR2(128) NOT NULL	The name of an ORACLE object to exclude from dropping.
OBJECT_TYPE	VARCHAR2(20) NOT NULL	The type of the ORACLE object (TABLE, INDEX, etc.) to exclude from dropping.
ACTIVE_FLAG	CHAR(1)	(Default: undefined, NULL) If the value is **Y** then this row entry is ACTIVE and the specified object will indeed be excluded of any drop action.
DATE_CREAT	DATE NOT NULL	The date this entry was created
USER_CREAT	VARCHAR2(128) NOT NULL	The user who created this entry
DATE_MODIF	DATE NOT NULL	The last time this row was modified by **USER_MODIF** user
USER_MODIF	VARCHAR2(128) NOT NULL	The user who last modified this row

6.1.15. DPP_PARAMETERS

This table contains parameters that can be configured based on the application context.

Column name	Type	Description
PRR_NAME	VARCHAR2(30) NOT NULL	Parameter name
PRR_VALUE	VARCHAR2(1000) NOT NULL	Parameter value
DESCR_ENG	VARCHAR2(2000) NOT NULL	Description in the English language
DESCR_FRA	VARCHAR2(2000) NOT NULL	Description in the French language
ITE_NAME	VARCHAR2(128) NOT NULL	Database instance name

Column name	Type	Description
DATE_CREAT	DATE NOT NULL	The date this entry was created
USER_CREAT	VARCHAR2(128) NOT NULL	The user who created this entry
DATE_MODIF	DATE NOT NULL	The last time this row was modified by **USER_MODIF** user
USER_MODIF	VARCHAR2(128) NOT NULL	The user who last modified this row

6.2. PL/SQL code

The implementation of the DPP Utility is made of several PL/SQL packages. All package names are prefixed with **DPP_** identifying them as pieces of the DPP Utility.

6.2.1. DPP_JOB_KRN package

This is the kernel of the DPP Utility and it provides the user with the API of the utility.

Those are the most important routines of the package:

6.2.1.1. export_schema

This procedure exports a schema by its functional name as defined in the **dpp_schemas** table. The use of the procedure is described in the chapter about the schema export.

A function with the same name and same purpose exists too.

This procedure must be used instead of the deprecated **export_logical_name** one, that has been removed starting from the version 24.2.

Format:

```
PROCEDURE export_schema(
   p_functional_name    IN VARCHAR2
 , p_options            IN VARCHAR2 DEFAULT NULL
);
```

Parameters:

- **p_functional_name:** Specifies the functional name of the schema to be exported.
- **p_options:** Optional parameters to control the export behavior.

6.2.1.2. import_schema

Imports a previously exported data file via the **export_schema** procedure. This procedure will raise an error if you try to import data into any schema on the production instance. A function with same name returns the status of the process. The use of this procedure is described in the chapter about the schema import.

This procedure must be used instead of the deprecated **import_logical_name** one, that has been removed starting from the version 24.2.

Format:

```
PROCEDURE import_schema(
   p_src_functional_name        IN VARCHAR2
 , p_trg_functional_name        IN VARCHAR2
 , p_options                    IN VARCHAR2 DEFAULT NULL
);
```

Parameters:

- **p_src_functional_name:** Specifies the functional name of the schema from which data was exported previously by a call to **export_schema**. This name must have been defined in the **dpp_schemas** table. The name is optionally appended with a date string in the **YYYYMMDD** format, to import a specific file. If the date clause is omitted, it defaults to the current date.
- **p_trg_functional_name:** Specifies the functional name of the schema into which data will be imported. This name must have been defined in the **dpp_schemas** table.
- **p_options:** Optional parameters to control the import behavior.

6.2.1.3. remove_files_older

This routine will remove all export dump files made ON AND BEFORE a particular date for a particular schema. The use of this procedure is described in the Removing old dump files chapter.

Format:

```
PROCEDURE remove_files_older(
   p_sma_name       IN VARCHAR2
 , p_date           IN DATE
);
```

Parameters:

- **p_sma_name:** Specifies the functional schema from which data was exported previously by a call to **export_schema**.
- **p_date**: Specifies the date for which all OS-files OLDER or EQUAL to this date will be removed from the file system.

6.2.1.4. remove_files_old_functional

This routine will remove ALL export dump files made BEFORE a particular date for a particular schema identified by its functional name. The use of this procedure is described in the Removing old dump files chapter.

Format:

```
PROCEDURE remove_files_old_functional(
   p_src_functional IN VARCHAR2
 , p_date           IN DATE
);
```

Parameters:

- **p_src_functional:** Specifies the functional schema from which data was exported previously by a call to **export_schema**.
- **p_date:** Specifies the date for which all OS-files OLDER or EQUAL to this date will me remove from the file system.

6.2.1.5. transfer_dumpfiles

This routine will transfer exported files from the one environment to another over a network connection. As previously mentioned, these environments are physically separated.

Warning: this routine needs a database link, pointing to an **APP_DPP_X** schema located on the other side. It assumes that the in/out directories have the same name in all environments. A function with same name returns the status of the process.

The use of the procedure is described in the Transferring dump file chapter.

Format:

```
PROCEDURE transfer_dumpfiles(
   p_schema     IN VARCHAR2
 , p_db_link    IN VARCHAR2
);
```

Parameters:

- **p_schema:** Specifies the functional schema from which data that is previously exported by a call to **export_schema**.
- **p_db_link:** Specifies the name of the database link to be used to transfer the files on the other side.

6.2.2. DPP_JOB_MEM package

This package stores and manages a list of **dpp_parameters** table rows in memory.

The routines of this package are described down below.

6.2.2.1. flush_prr

This routine flushes a table in memory containing the utility parameters.

Format:

```
PROCEDURE flush_prr;
```

6.2.2.2. load_prr

This routine loads a table in memory with the utility parameters.

Format:

```
PROCEDURE load_prr;
```

6.2.2.3. get_prr

This routine retrieves a parameter row from the table in memory containing the utility parameters.

Format:

```
FUNCTION get_prr(
   p_prr_name       IN dpp_parameters.prr_name%TYPE
 , p_ite_name       IN dpp_parameters.ite_name%TYPE := NULL
) RETURN dpp_parameters%ROWTYPE;
```

Parameters:

- **p_prr_name:** The name of the parameter.
- **p_ite_name:** database instance name

6.2.3. DPP_JOB_VAR package

This package contains global variables and constants for the **dpp_job_krn** package, to make this one stateless.

6.2.4. DPP_INJ_KRN package

This package provides injection services to the **dpp_job_krn** main package. These services allow injecting procedures in the target schema which will perform object "drops" or "creations" such as tables, views, sequences …. including the export and import jobs.

6.2.5. DPP_INJ_VAR package

This package contains global variables and constants for the **dpp_inj_krn** package, to make this one stateless.

6.2.6. DPP_ITF_KRN package

This package provides services to log the export/import transfer actions in the **dpp_job_runs** and **dpp_job_logs**.

6.2.7. DPP_ITF_VAR package

It contains global variables and constants for the **dpp_itf_krn** package, to make this one stateless.

6.2.8. DPP_UTILITY_LIC package

It contains the licence terms.

6.2.9. MAIL_UTILITY_KRN package

This package is used by the DPP Utility but is part of the MAIL Utility.

This package provides email services. The use of this package is described in the Sending mail chapter.

6.2.10. MAIL_UTILITY_VAR package

This package is used by the DPP Utility but is part of the MAIL Utility.

It contains global variables and constants for the **mail_utility_krn** package, to make this one stateless.

6.2.11. DPP_MONITORING_KRN package

This package implements the kernel of the timeout monitoring.

The routines of this package are described down below.

6.2.11.1. exec_monitoring

This procedure is automatically called by the **DPP_MONITORING** monitoring job every five minutes and implements the timeout detection.

Format:

```
PROCEDURE exec_monitoring(p_debug IN BOOLEAN := FALSE);
```

Parameters:

- **p_debug:** Whether the procedure must be executed in debug mode that generates execution messages.

6.2.12. DPP_MONITORING_VAR package

This package contains the variables and constants used by the **dpp_monitoring_krn** package in order to make it stateless.

6.2.13. DPP_SEC_VAR package

The secret constants are declared in this package, like the **SEC Utility** encryption key. This package is compiled making those constant values unreadable.

6.2.14. DPP_CNF_KRN package

This package implements the API that manages the configuration data. It allows to create and maintain the configuration data without having to insert or modify data directory in the database tables.

6.2.14.1. create_database_link

This procedure creates a database link in a schema managed by the DPP Utility.

Oracle currently does not allow a database user to create a database link in another schema. As workaround, this procedure creates a temporary stored procedure in the schema the database link must be created in. The database link is actually created by this temporary stored procedure. The stored procedure is deleted once the database link is created. The database user needs then to be granted with the needed privileges to create, execute and drop a stored procedure in another schema:

- **create any procedure**
- **execute any procedure**
- **drop any procedure**

Format:

```
PROCEDURE create_database_link (
   p_target_schema         IN    VARCHAR2
 , p_db_link_name          IN    VARCHAR2
 , p_db_link_user          IN    VARCHAR2
 , p_db_link_pwd           IN    VARCHAR2
 , p_db_link_conn_string   IN    VARCHAR2
);
```

Parameters:

- **p_target_schema:** name of the schema the database link should be created in
- **p_db_link_name:** name of the database link to be created
- **p_db_link_user:** user ID or schema the database link points to
- **p_db_link_pwd:** database link password
- **p_db_link_conn_string:** database link connection string

6.2.14.2. drop_database_link

This procedure drops a database link in a schema managed by the DPP Utility.

Oracle currently does not allow a database user to drop a database link in another schema. As workaround, this procedure creates a temporary stored procedure in the schema the database link was created in. The database link is actually dropped by this temporary stored procedure. The stored procedure is deleted once the database link is dropped. The database user needs then to be granted with the needed privileges to create, execute and drop a stored procedure in another schema:

- **create any procedure**
- **execute any procedure**
- **drop any procedure**

Format:

```
PROCEDURE drop_database_link (
   p_target_schema         IN    VARCHAR2
 , p_db_link_name          IN    VARCHAR2
);
```

Parameters:

- **p_target_schema:** name of the schema the database link was created in
- **p_db_link_name:** name of the database link to be deleted

6.2.14.3. insert_instance

This procedure inserts a new database instance in the **dpp_instances** table.

Format:

```
PROCEDURE insert_instance(
   p_instance_name         IN  dpp_instances.ite_name%TYPE
 , p_descr_eng             IN  dpp_instances.descr_eng%TYPE
 , p_descr_fra             IN  dpp_instances.descr_fra%TYPE
 , p_production_flag       IN  dpp_instances.production_flag%TYPE := NULL
 , p_env_name              IN  dpp_instances.env_name%TYPE
 , p_date_creat            IN  dpp_instances.date_creat%TYPE      := NULL
 , p_user_creat            IN  dpp_instances.user_creat%TYPE      := NULL
 , p_date_modif            IN  dpp_instances.date_modif%TYPE      := NULL
 , p_user_modif            IN  dpp_instances.user_modif%TYPE      := NULL
);
```

Parameters:

- **p_instance_name:** name of the database instance
- **p_descr_eng:** description in English
- **p_descr_fra:** description in French
- **p_production_flag:** whether this is a production instance
- **p_env_name:** environment name
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.4. delete_instance

This procedure deletes a database instance from the **dpp_instances** table.

Format:

```
PROCEDURE delete_instance(
   p_instance_name         IN  dpp_instances.ite_name%TYPE
);
```

Parameters:

- **p_instance_name:** name of the database instance to be deleted.

6.2.14.5. update_instance

This procedure updates a database instance in the **dpp_instances** table.

Format:

```
PROCEDURE update_instance(
   p_instance_name         IN  dpp_instances.ite_name%TYPE
 , p_descr_eng             IN  dpp_instances.descr_eng%TYPE       := NULL
 , p_descr_fra             IN  dpp_instances.descr_fra%TYPE       := NULL
 , p_production_flag       IN  dpp_instances.production_flag%TYPE := NULL
 , p_env_name              IN  dpp_instances.env_name%TYPE        := NULL
 , p_date_creat            IN  dpp_instances.date_creat%TYPE      := NULL
 , p_user_creat            IN  dpp_instances.user_creat%TYPE      := NULL
 , p_date_modif            IN  dpp_instances.date_modif%TYPE      := NULL
 , p_user_modif            IN  dpp_instances.user_modif%TYPE      := NULL
);
```

Parameters:

- **p_instance_name:** name of the database instance
- **p_descr_eng:** description in English
- **p_descr_fra:** description in French
- **p_production_flag:** whether this is a production instance
- **p_env_name:** environment name
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.6. insert_schema_type

This procedure inserts a new schema type in the **dpp_schema_types** table.

Format:

```
PROCEDURE insert_schema_type(
   p_schema_type_name      IN  dpp_schema_types.ste_name%TYPE
 , p_date_creat            IN  dpp_schema_types.date_creat%TYPE   := NULL
 , p_user_creat            IN  dpp_schema_types.user_creat%TYPE   := NULL
 , p_date_modif            IN  dpp_schema_types.date_modif%TYPE   := NULL
 , p_user_modif            IN dpp_schema_types.user_modif%TYPE    := NULL
);
```

Parameters:

- **p_schema_type_name:** name of the schema type to be created
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.7. delete_schema_type

This procedure deletes a schema type from the **dpp_schema_types** table.

Format:

```
PROCEDURE delete_schema_type(
   p_schema_type_name      IN  dpp_schema_types.ste_name%TYPE
);
```

Parameters:

- **p_schema_type_name:** name of the schema type to be deleted

6.2.14.8. update_schema_type

This procedure updates a schema type in the **dpp_schema_types** table.

Format:

```
PROCEDURE update_schema_type(
   p_schema_type_name      IN  dpp_schema_types.ste_name%TYPE
 , p_date_creat            IN  dpp_schema_types.date_creat%TYPE   := NULL
 , p_user_creat            IN  dpp_schema_types.user_creat%TYPE   := NULL
 , p_date_modif            IN  dpp_schema_types.date_modif%TYPE   := NULL
 , p_user_modif            IN  dpp_schema_types.user_modif%TYPE   := NULL
);
```

Parameters:

- **p_schema_type_name:** name of the schema type to be updated
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.9. insert_role

This procedure inserts a new role in the **dpp_roles** table.

Format:

```
PROCEDURE insert_role(
   p_role_name             IN  dpp_roles.rle_name%TYPE
 , p_descr_eng             IN  dpp_roles.descr_eng%TYPE
 , p_descr_fra             IN  dpp_roles.descr_fra%TYPE
 , p_date_creat            IN  dpp_roles.date_creat%TYPE       := NULL
 , p_user_creat            IN  dpp_roles.user_creat%TYPE       := NULL
 , p_date_modif            IN  dpp_roles.date_modif%TYPE       := NULL
 , p_user_modif            IN  dpp_roles.user_modif%TYPE       := NULL
);
```

Parameters:

- **p_role_name:** name of the role to be created
- **p_descr_eng:** description in English
- **p_descr_fra:** description in French
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.10. delete_role

This procedure deletes a role from the **dpp_roles** table.

Format:

```
PROCEDURE delete_role(
   p_role_name             IN  dpp_roles.rle_name%TYPE
);
```

Parameters:

- **p_role_name:** name of the role to be deleted

6.2.14.11. update_role

This procedure updates a role in the **dpp_roles** table.

Format:

```
PROCEDURE update_role(
   p_role_name             IN  dpp_roles.rle_name%TYPE
 , p_descr_eng             IN  dpp_roles.descr_eng%TYPE        := NULL
 , p_descr_fra             IN  dpp_roles.descr_fra%TYPE        := NULL
 , p_date_creat            IN  dpp_roles.date_creat%TYPE       := NULL
 , p_user_creat            IN  dpp_roles.user_creat%TYPE       := NULL
 , p_date_modif            IN  dpp_roles.date_modif%TYPE       := NULL
 , p_user_modif            IN  dpp_roles.user_modif%TYPE       := NULL
);
```

Parameters:

- **p_role_name:** name of the role to be updated
- **p_descr_eng:** description in English
- **p_descr_fra:** description in French
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.12. insert_schema

This function inserts a new database schema in the **dpp_schemas** table. If the schema ID is not passed as parameter, it is computed by the function.

Format:

```
FUNCTION insert_schema(
   p_schema_id             IN  dpp_schemas.sma_id%TYPE            := NULL
 , p_instance_name         IN  dpp_schemas.ite_name%TYPE
 , p_role_name             IN  dpp_schemas.rle_name%TYPE          := NULL
 , p_schema_type_name      IN  dpp_schemas.ste_name%TYPE          := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE
 , p_schema_name           IN  dpp_schemas.sma_name%TYPE
 , p_production_flag       IN  dpp_schemas.production_flag%TYPE   := NULL
 , p_date_from             IN  dpp_schemas.date_from%TYPE         := NULL
 , p_date_to               IN  dpp_schemas.date_to%TYPE           := NULL
 , p_date_creat            IN  dpp_schemas.date_creat%TYPE        := NULL
 , p_user_creat            IN  dpp_schemas.user_creat%TYPE        := NULL
 , p_date_modif            IN  dpp_schemas.date_modif%TYPE        := NULL
 , p_user_modif            IN  dpp_schemas.user_modif%TYPE        := NULL
)
RETURN dpp_schemas.sma_id%TYPE;
```

Parameters:

- **p_schema_id:** ID of the schema to be created
- **p_instance_name:** database instance name
- **p_role_name:** role name
- **p_schema_type_name:** schema type name
- **p_functional_name:** functional name
- **p_schema_name:** schema name
- **p_production_flag:** whether this is a production schema
- **p_date_from:** start date of the validity period
- **p_date_to:** end date of the validity period
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

Return: ID of the new schema

6.2.14.13. delete_schema

This procedure deletes a database schema from the **dpp_schemas** table. The schema to be deleted can be identified by its ID or by its functional name which is unique.

Format:

```
PROCEDURE delete_schema(
   p_schema_id             IN  dpp_schemas.sma_id%TYPE            := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE   := NULL
);
```

Parameters:

- **p_schema_id:** ID of the schema to be deleted
- **p_functional_name:** functional name of the schema to be deleted

6.2.14.14. update_schema

This procedure updates a database schema in the **dpp_schemas** table.

Format:

```
PROCEDURE update_schema(
   p_schema_id             IN  dpp_schemas.sma_id%TYPE
 , p_instance_name         IN  dpp_schemas.ite_name%TYPE          := NULL
 , p_role_name             IN  dpp_schemas.rle_name%TYPE          := NULL
 , p_schema_type_name      IN  dpp_schemas.ste_name%TYPE          := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE   := NULL
 , p_schema_name           IN  dpp_schemas.sma_name%TYPE          := NULL
 , p_production_flag       IN  dpp_schemas.production_flag%TYPE   := NULL
 , p_date_from             IN  dpp_schemas.date_from%TYPE         := NULL
 , p_date_to               IN  dpp_schemas.date_to%TYPE           := NULL
 , p_date_creat            IN  dpp_schemas.date_creat%TYPE        := NULL
 , p_user_creat            IN  dpp_schemas.user_creat%TYPE        := NULL
 , p_date_modif            IN  dpp_schemas.date_modif%TYPE        := NULL
 , p_user_modif            IN  dpp_schemas.user_modif%TYPE        := NULL
);
```

Parameters:

- **p_schema_id:** ID of the schema to be update
- **p_instance_name:** database instance name
- **p_role_name:** role name
- **p_schema_type_name:** schema type name
- **p_functional_name:** functional name
- **p_schema_name:** schema name
- **p_production_flag:** whether this is a production schema
- **p_date_from:** start date of the validity period
- **p_date_to:** end date of the validity period
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.15. duplicate_schema

This function duplicates a database schema in the **dpp_schemas** table. If the target schema ID is not passed as parameter, it is computed by the function. The source schema can be identified by its ID or its functional name.

Format:

```
FUNCTION duplicate_schema(
   p_src_schema_id         IN  dpp_schemas.sma_id%TYPE            := NULL
 , p_src_functional_name   IN  dpp_schemas.functional_name%TYPE   := NULL
 , p_trg_schema_id         IN  dpp_schemas.sma_id%TYPE            := NULL
 , p_trg_functional_name   IN  dpp_schemas.functional_name%TYPE
 , p_date_creat            IN  dpp_schemas.date_creat%TYPE        := NULL
 , p_user_creat            IN  dpp_schemas.user_creat%TYPE        := NULL
 , p_date_modif            IN  dpp_schemas.date_modif%TYPE        := NULL
 , p_user_modif            IN  dpp_schemas.user_modif%TYPE        := NULL
) RETURN dpp_schemas.sma_id%TYPE;
```

Parameters:

- **p_src_schema_id:** ID of the source schema
- **p_src_functional_name:** functional name of the source schema
- **p_trg_schema_id:** ID of the target schema
- **p_trg_functional_name:** functional name of the target schema
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

Return: ID of the new schema

6.2.14.16. insert_nodrop_object

This procedure inserts a new item in the table of objects that should never be dropped (**dpp_nodrop_objects** table). The schema can be identified by its ID or by its functional name which is unique.

Format:

```
PROCEDURE insert_nodrop_object(
   p_schema_id             IN  dpp_nodrop_objects.sma_id%TYPE        := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE      := NULL
 , p_object_name           IN  dpp_nodrop_objects.object_name%TYPE
 , p_object_type           IN  dpp_nodrop_objects.object_type%TYPE
 , p_active_flag           IN  dpp_nodrop_objects.active_flag%TYPE
 , p_date_creat            IN  dpp_nodrop_objects.date_creat%TYPE    := NULL
 , p_user_creat            IN  dpp_nodrop_objects.user_creat%TYPE    := NULL
 , p_date_modif            IN  dpp_nodrop_objects.date_modif%TYPE    := NULL
 , p_user_modif            IN  dpp_nodrop_objects.user_modif%TYPE    := NULL
);
```

Parameters:

- **p_schema_id:** ID of the schema
- **p_functional_name:** functional name of the schema
- **p_object_name:** name of the object that should not be dropped
- **p_object_type:** type of the object
- **p_active_flag:** whether the rule is active
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.17. delete_nodrop_object

This procedure deletes an item from the table of objects that should never be dropped (**dpp_nodrop_objects** table). The schema can be identified by its ID or by its functional name which is unique.

Format:

```
PROCEDURE delete_nodrop_object(
   p_schema_id             IN  dpp_nodrop_objects.sma_id%TYPE        := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE      := NULL
 , p_object_name           IN  dpp_nodrop_objects.object_name%TYPE
 , p_object_type           IN  dpp_nodrop_objects.object_type%TYPE
);
```

Parameters:

- **p_schema_id:** ID of the schema
- **p_functional_name:** functional name of the schema
- **p_object_name:** name of the object that should not be dropped
- **p_object_type:** type of the object

6.2.14.18. update_nodrop_object

This procedure updates an item in the table of objects that should never be dropped (**dpp_nodrop_objects** table). The schema can be identified by its ID or by its functional name which is unique.

Format:

```
PROCEDURE update_nodrop_object(
   p_schema_id             IN  dpp_nodrop_objects.sma_id%TYPE        := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE      := NULL
 , p_object_name           IN  dpp_nodrop_objects.object_name%TYPE
 , p_object_type           IN  dpp_nodrop_objects.object_type%TYPE
 , p_active_flag           IN  dpp_nodrop_objects.active_flag%TYPE   := NULL
 , p_date_creat            IN  dpp_nodrop_objects.date_creat%TYPE    := NULL
 , p_user_creat            IN  dpp_nodrop_objects.user_creat%TYPE    := NULL
 , p_date_modif            IN  dpp_nodrop_objects.date_modif%TYPE    := NULL
 , p_user_modif            IN  dpp_nodrop_objects.user_modif%TYPE    := NULL
);
```

Parameters:

- **p_schema_id:** ID of the schema
- **p_functional_name:** functional name of the schema
- **p_object_name:** name of the object that should not be dropped
- **p_object_type:** type of the object
- **p_active_flag:** whether the rule is active
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.19. duplicate_nodrop_object

This procedure duplicates an item in the table of objects that should never be dropped (**dpp_nodrop_objects** table) from a schema to another one. The schemas can be identified by their ID or by their functional name which is unique.

Format:

```
PROCEDURE duplicate_nodrop_object(
   p_schema_id             IN  dpp_nodrop_objects.sma_id%TYPE     := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE   := NULL
 , p_object_name           IN  dpp_nodrop_objects.object_name%TYPE
 , p_object_type           in  dpp_nodrop_objects.object_type%TYPE
 , p_trg_schema_id         IN  dpp_nodrop_objects.sma_id%TYPE     := NULL
 , p_trg_functional_name   IN  dpp_schemas.functional_name%TYPE   := NULL
 , p_date_creat            IN  dpp_nodrop_objects.date_creat%TYPE := NULL
 , p_user_creat            IN  dpp_nodrop_objects.user_creat%TYPE := NULL
 , p_date_modif            IN  dpp_nodrop_objects.date_modif%TYPE := NULL
 , p_user_modif            IN  dpp_nodrop_objects.user_modif%TYPE := NULL
);
```

Parameters:

- **p_schema_id:** ID of the source schema
- **p_functional_name:** functional name of the source schema
- **p_object_name:** name of the object that should not be dropped
- **p_object_type:** type of the object

- **p_trg_schema_id:** ID of the target schema
- **p_trg_functional_name:** functional name of the target schema
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.20. insert_action

This procedure inserts a new action in the **dpp_actions** table. The schema can be identified by its ID or by its functional name which is unique.

Format:

```
PROCEDURE insert_action(
   p_schema_id             IN  dpp_actions.sma_id%TYPE            := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE   := NULL
 , p_atn_usage             IN  dpp_actions.atn_usage%TYPE
 , p_atn_type              IN  dpp_actions.atn_type%TYPE
 , p_exec_order            IN  dpp_actions.execution_order%TYPE
 , p_block_text            IN  dpp_actions.block_text%TYPE
 , p_active_flag           IN  dpp_actions.active_flag%TYPE
 , p_date_creat            IN  dpp_actions.date_creat%TYPE        := NULL
 , p_user_creat            IN  dpp_actions.user_creat%TYPE        := NULL
 , p_date_modif            IN  dpp_actions.date_modif%TYPE        := NULL
 , p_user_modif            IN  dpp_actions.user_modif%TYPE        := NULL
);
```

Parameters:

- **p_schema_id:** ID of the schema
- **p_functional_name:** functional name of the schema
- **p_atn_usage:** action usage
- **p_atn_type:** action type
- **p_exec_order:** execution order
- **p_block_text:** PL/SQL block to be executed
- **p_active_flag:** whether the action is active
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.21. insert_http_request_action

This procedure inserts a new action in the **dpp_actions** table. When triggered, this action sends an HTTP request to the URL passed as parameter. The schema can be identified by its ID or by its functional name which is unique.

Format:

```
PROCEDURE insert_http_request_action(
   p_schema_id             IN  dpp_actions.sma_id%TYPE            := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE   := NULL
 , p_atn_usage             IN  dpp_actions.atn_usage%TYPE
 , p_atn_type              IN  dpp_actions.atn_type%TYPE
 , p_exec_order            IN  dpp_actions.execution_order%TYPE
 , p_url                   IN  VARCHAR2
 , p_wallet_path           IN  VARCHAR2                           := NULL
 , p_proxy                 IN  VARCHAR2                           := NULL
 , p_active_flag           IN  dpp_actions.active_flag%TYPE
 , p_date_creat            IN  dpp_actions.date_creat%TYPE        := NULL
 , p_user_creat            IN  dpp_actions.user_creat%TYPE        := NULL
 , p_date_modif            IN  dpp_actions.date_modif%TYPE        := NULL
 , p_user_modif            IN  dpp_actions.user_modif%TYPE        := NULL
);
```

Parameters:

- **p_schema_id:** ID of the schema
- **p_functional_name:** functional name of the schema
- **p_atn_usage:** action usage
- **p_atn_type:** action type
- **p_exec_order:** execution order
- **p_url:** the URL the HTTP request must be sent to
- **p_wallet_path:** if a certificate is needed to send the request, the path of the wallet the certificate is stored in must be mentioned in this parameter
- **p_proxy:** proxy server address
- **p_active_flag:** whether the action is active
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

The PL/SQL code of the action is automatically generated by this function. The HTTP request will be sent using the HTTP Utility. The PL/SQL code that is generated is based on the following template:

```
BEGIN
   http_utility_krn.send_http_request(
      '{url}'
    , '{wallet_path}'
    , '{proxy}'
   );
END;
```

Where:

- *{url}* is replaced with the value of *p_url*.
- *{wallet_path}* is replaced with the value of *p_wallet_path*.
- *{proxy}* is replaced with the value of *p_proxy*.

6.2.14.22. insert_http_request_action_enc

This procedure inserts a new action in the **dpp_actions** table. When triggered, this action sends an HTTP request to the URL passed as parameter. The schema can be identified by its ID or by its functional name which is unique.

The difference with the insert_http_request_action procedure is that this version of the procedure encrypts the URL and the wallet path using the SEC Utility.

Format:

```
PROCEDURE insert_http_request_action_enc(
   p_schema_id             IN  dpp_actions.sma_id%TYPE            := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE   := NULL
 , p_atn_usage             IN  dpp_actions.atn_usage%TYPE
 , p_atn_type              IN  dpp_actions.atn_type%TYPE
 , p_exec_order            IN  dpp_actions.execution_order%TYPE
 , p_url                   IN  VARCHAR2
 , p_wallet_path           IN  VARCHAR2                           := NULL
 , p_proxy                 IN  VARCHAR2                           := NULL
 , p_active_flag           IN  dpp_actions.active_flag%TYPE
 , p_date_creat            IN  dpp_actions.date_creat%TYPE        := NULL
 , p_user_creat            IN  dpp_actions.user_creat%TYPE        := NULL
 , p_date_modif            IN  dpp_actions.date_modif%TYPE        := NULL
 , p_user_modif            IN  dpp_actions.user_modif%TYPE        := NULL
 , p_ph_suffix             IN  VARCHAR2
);
```

Parameters:

- **p_schema_id:** ID of the schema
- **p_functional_name:** functional name of the schema
- **p_atn_usage:** action usage
- **p_atn_type:** action type
- **p_exec_order:** execution order
- **p_url:** the URL the HTTP request must be sent to
- **p_wallet_path:** if a certificate is needed to send the request, the path of the wallet the certificate is stored in must be mentioned in this parameter
- **p_proxy:** proxy server address
- **p_active_flag:** whether the action is active
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID
- **p_ph_suffix:** the URL and the wallet path are encrypted by the procedure using the SEC Utility. The encrypted values are stored in the tables of the utility using the **URL** and **WALLET_PATH** variables followed by this suffix. This allows to have several actions using different URL's and wallet paths.

The PL/SQL code of the action is automatically generated by this function and the values of *p_url* and *p_wallet_path* are automatically encrypted. The HTTP request will be sent using the HTTP Utility. The PL/SQL code that is generated is based on the following template:

```
BEGIN
   http_utility_krn.send_http_request(
      '{url}'
    , '{wallet_path}'
    , '{proxy}'
   );
END;
```

Where:

- *{url}* is replaced with the encrypted value of *p_url*.
- *{wallet_path}* is replaced with the encrypted value of *p_wallet_path*.
- *{proxy}* is replaced with the value of *p_proxy*.

6.2.14.23. insert_http_gitlab_req_action

This procedure inserts a new action in the **dpp_actions** table. When triggered, this action sends an HTTP request to a GitLab server to trigger a pipeline or a pipeline job. The URL root, the pipeline token, the wallet path, the proxy server and a gitLab variable to be initialized are passed as parameters. The schema can be identified by its ID or by its functional name which is unique.

Format:

```
PROCEDURE insert_http_gitlab_req_action(
   p_schema_id             IN  dpp_actions.sma_id%TYPE            := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE   := NULL
 , p_atn_usage             IN  dpp_actions.atn_usage%TYPE
 , p_atn_type              IN  dpp_actions.atn_type%TYPE
 , p_exec_order            IN  dpp_actions.execution_order%TYPE
 , p_url                   IN  VARCHAR2
 , p_wallet_path           IN  VARCHAR2                           := NULL
 , p_proxy                 IN  VARCHAR2                           := NULL
 , p_token                 IN  VARCHAR2                           := NULL
 , p_var_name              IN  VARCHAR2                           := NULL
 , p_var_value             IN  VARCHAR2                           := NULL
 , p_active_flag           IN  dpp_actions.active_flag%TYPE
 , p_date_creat            IN  dpp_actions.date_creat%TYPE        := NULL
 , p_user_creat            IN  dpp_actions.user_creat%TYPE        := NULL
 , p_date_modif            IN  dpp_actions.date_modif%TYPE        := NULL
 , p_user_modif            IN  dpp_actions.user_modif%TYPE        := NULL
);
```

Parameters:

- **p_schema_id:** ID of the schema
- **p_functional_name:** functional name of the schema
- **p_atn_usage:** action usage
- **p_atn_type:** action type
- **p_exec_order:** execution order
- **p_url:** the root of the URL the HTTP request must be sent to
- **p_wallet_path:** if a certificate is needed to send the request, the path of the wallet the certificate is stored in must be mentioned in this parameter
- **p_proxy:** proxy server address
- **p_token:** the GitLab pipeline trigger token
- **p_var_name:** the name of a GitLab variable to be initialized
- **p_var_value:** the value if a GitLab variable to be initialized
- **p_active_flag:** whether the action is active
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID

- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

The PL/SQL code of the action is automatically generated by this function. The HTTP request will be sent using the HTTP Utility. The PL/SQL code that is generated is based on the following template:

```
BEGIN
   http_utility_krn.send_gitlab_http_request(
      '{url}'
    , '{token}'
    , '{var_name}'
    , '{var_value}'
    , '{wallet_path}'
    , '{proxy}'
   );
END;
```

Where:

- *{url}* is replaced with the value of *p_url*.
- *{token}* is replaced with the value of *p_token*.
- *{var_name}* is replaced with the value of *p_var_name*.
- *{var_value}* is replaced with the value of *p_var_value*.
- *{wallet_path}* is replaced with the value of *p_wallet_path*.
- *{proxy}* is replaced with the value of *p_proxy*.

6.2.14.24. insert_http_gitlab_req_action_enc

This procedure inserts a new action in the **dpp_actions** table. When triggered, this action sends an HTTP request to a GitLab server to trigger a pipeline or a pipeline job. The URL root, the pipeline token, the wallet path, the proxy server and a gitLab variable to be initialized are passed as parameters. The schema can be identified by its ID or by its functional name which is unique.

The difference with the insert_http_gitlab_req_action procedure is that this version of the procedure encrypts the URL, the token and the wallet path using the SEC Utility.

Format:

```
PROCEDURE insert_http_gitlab_req_action_enc(
   p_schema_id             IN  dpp_actions.sma_id%TYPE            := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE   := NULL
 , p_atn_usage             IN  dpp_actions.atn_usage%TYPE
 , p_atn_type              IN  dpp_actions.atn_type%TYPE
 , p_exec_order            IN  dpp_actions.execution_order%TYPE
 , p_url                   IN  VARCHAR2
 , p_wallet_path           IN  VARCHAR2                           := NULL
 , p_proxy                 IN  VARCHAR2                           := NULL
 , p_token                 IN  VARCHAR2                           := NULL
 , p_var_name              IN  VARCHAR2                           := NULL
 , p_var_value             IN  VARCHAR2                           := NULL
 , p_active_flag           IN  dpp_actions.active_flag%TYPE
 , p_date_creat            IN  dpp_actions.date_creat%TYPE        := NULL
 , p_user_creat            IN  dpp_actions.user_creat%TYPE        := NULL
 , p_date_modif            IN  dpp_actions.date_modif%TYPE        := NULL
 , p_user_modif            IN  dpp_actions.user_modif%TYPE        := NULL
 , p_ph_suffix             IN  VARCHAR2
);
```

Parameters:

- **p_schema_id:** ID of the schema
- **p_functional_name:** functional name of the schema
- **p_atn_usage:** action usage
- **p_atn_type:** action type
- **p_exec_order:** execution order
- **p_url:** the root of the URL the HTTP request must be sent to
- **p_wallet_path:** if a certificate is needed to send the request, the path of the wallet the certificate is stored in must be mentioned in this parameter
- **p_proxy:** proxy server address
- **p_token:** the GitLab pipeline trigger token
- **p_var_name:** the name of a GitLab variable to be initialized
- **p_var_value:** the value if a GitLab variable to be initialized
- **p_active_flag:** whether the action is active
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID
- **p_ph_suffix:** the URL, the GitLab pipeline trigger token and the wallet path are encrypted by the procedure using the SEC Utility. The encrypted values are stored in the tables of the utility using the **URL**, **TOKEN** and **WALLET_PATH** variables followed by this suffix. This allows to have several actions using different URL's, tokens and wallet paths.

The PL/SQL code of the action is automatically generated by this function and the values of *p_url*, *p_token* and *p_wallet_path* are automatically encrypted. The HTTP request will be sent using the HTTP Utility. The PL/SQL code that is generated is based on the following template:

```
BEGIN
   http_utility_krn.send_gitlab_http_request(
      '{url}'
    , '{token}'
    , '{var_name}'
    , '{var_value}'
    , '{wallet_path}'
    , '{proxy}'
   );
END;
```

Where:

- *{url}* is replaced with the encrypted value of *p_url*.
- *{token}* is replaced with the encrypted value of *p_token*.
- *{var_name}* is replaced with the value of *p_var_name*.
- *{var_value}* is replaced with the value of *p_var_value*.
- *{wallet_path}* is replaced with the encrypted value of *p_wallet_path*.
- *{proxy}* is replaced with the value of *p_proxy*.

6.2.14.25. delete_action

This procedure deletes an action from **dpp_actions** table. The schema can be identified by its ID or by its functional name which is unique.

Format:

```
PROCEDURE delete_action(
   p_schema_id             IN  dpp_actions.sma_id%TYPE            := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE   := NULL
 , p_atn_usage             IN  dpp_actions.atn_usage%TYPE
 , p_atn_type              IN  dpp_actions.atn_type%TYPE
 , p_exec_order            IN  dpp_actions.execution_order%TYPE
);
```

Parameters:

- **p_schema_id:** ID of the schema
- **p_functional_name:** functional name of the schema
- **p_atn_usage:** action usage
- **p_atn_type:** action type
- **p_exec_order:** execution order

6.2.14.26. update_action

This procedure updates an action in the **dpp_actions** table. The schema can be identified by its ID or by its functional name which is unique.

Format:

```
PROCEDURE update_action(
   p_schema_id             IN  dpp_actions.sma_id%TYPE            := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE   := NULL
 , p_atn_usage             IN  dpp_actions.atn_usage%TYPE
 , p_atn_type              IN  dpp_actions.atn_type%TYPE
 , p_exec_order            IN  dpp_actions.execution_order%TYPE
 , p_block_text            IN  dpp_actions.block_text%TYPE        := NULL
 , p_active_flag           IN  dpp_actions.active_flag%TYPE       := NULL
 , p_date_creat            IN  dpp_actions.date_creat%TYPE        := NULL
 , p_user_creat            IN  dpp_actions.user_creat%TYPE        := NULL
 , p_date_modif            IN  dpp_actions.date_modif%TYPE        := NULL
 , p_user_modif            IN  dpp_actions.user_modif%TYPE        := NULL
);
```

Parameters:

- **p_schema_id:** ID of the schema
- **p_functional_name:** functional name of the schema
- **p_atn_usage:** action usage
- **p_atn_type:** action type
- **p_exec_order:** execution order
- **p_block_text:** PL/SQL block to be executed
- **p_active_flag:** whether the action is active
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.27. update_action_exec_order

This procedure updates the execution order of an action in the **dpp_actions** table. The schema can be identified by its ID or by its functional name which is unique.

Format:

```
PROCEDURE update_action_exec_order(
   p_schema_id             IN  dpp_actions.sma_id%TYPE            := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE   := NULL
 , p_atn_usage             IN  dpp_actions.atn_usage%TYPE
 , p_atn_type              IN  dpp_actions.atn_type%TYPE
 , p_curr_exec_order       IN  dpp_actions.execution_order%TYPE
 , p_new_exec_order        IN  dpp_actions.execution_order%TYPE
 , p_date_modif            IN  dpp_actions.date_modif%TYPE        := NULL
 , p_user_modif            IN  dpp_actions.user_modif%TYPE        := NULL
);
```

Parameters:

- **p_schema_id:** ID of the schema
- **p_functional_name:** functional name of the schema
- **p_atn_usage:** action usage
- **p_atn_type:** action type
- **p_curr_exec_order:** current execution order
- **p_new_exec_order:** new execution order
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.28. duplicate_action

This procedure duplicates an action in the **dpp_actions** table from a schema to another none. The schemas can be identified by their ID or by their functional name which is unique.

Format:

```
PROCEDURE duplicate_action(
   p_schema_id             IN  dpp_actions.sma_id%TYPE            := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE   := NULL
 , p_atn_usage             IN  dpp_actions.atn_usage%TYPE
 , p_atn_type              IN  dpp_actions.atn_type%TYPE
 , p_exec_order            IN  dpp_actions.execution_order%TYPE
 , p_trg_schema_id         IN  dpp_actions.sma_id%TYPE            := NULL
 , p_trg_functional_name   IN  dpp_schemas.functional_name%TYPE   := NULL
 , p_date_creat            IN  dpp_actions.date_creat%TYPE        := NULL
 , p_user_creat            IN  dpp_actions.user_creat%TYPE        := NULL
 , p_date_modif            IN  dpp_actions.date_modif%TYPE        := NULL
 , p_user_modif            IN  dpp_actions.user_modif%TYPE        := NULL
);
```

Parameters:

- **p_schema_id:** ID of the schema
- **p_functional_name:** functional name of the schema
- **p_atn_usage:** action usage
- **p_atn_type:** action type
- **p_exec_order:** execution order
- **p_trg_schema_id:** ID of the target schema
- **p_trg_functional_name:** functional name of the target schema
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.29. insert_parameter

This procedure inserts a new parameter in the **dpp_parameters** table. When the database instance name is not passed as argument, the new parameter is common to all database instances.

Format:

```
PROCEDURE insert_parameter(
   p_param_name            IN  dpp_parameters.prr_name%TYPE
 , p_param_value           IN  dpp_parameters.prr_value%TYPE
 , p_descr_eng             IN  dpp_parameters.descr_eng%TYPE
 , p_descr_fra             IN  dpp_parameters.descr_fra%TYPE
 , p_instance_name         IN  dpp_parameters.ite_name%TYPE         := NULL
 , p_date_creat            IN  dpp_parameters.date_creat%TYPE       := NULL
 , p_user_creat            IN  dpp_parameters.user_creat%TYPE       := NULL
 , p_date_modif            IN  dpp_parameters.date_modif%TYPE       := NULL
 , p_user_modif            IN  dpp_parameters.user_modif%TYPE       := NULL
);
```

Parameters:

- **p_param_name:** parameter name
- **p_param_value:** parameter value
- **p_descr_eng:** description in English
- **p_descr_fra:** description in French
- **p_instance_name:** database instance name
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.30. delete_parameter

This procedure deletes a from the **dpp_parameters** table.

Format:

```
PROCEDURE delete_parameter(
   p_param_name            IN  dpp_parameters.prr_name%TYPE
 , p_instance_name         IN  dpp_parameters.ite_name%TYPE         := NULL
);
```

Parameters:

- **p_param_name:** parameter name
- **p_instance_name:** database instance name

6.2.14.31. update_parameter

This procedure updates a parameter in the **dpp_parameters** table. When the database instance name is not passed as argument, the new parameter is common to all database instances.

Format:

```
PROCEDURE update_parameter(
   p_param_name            IN  dpp_parameters.prr_name%TYPE
 , p_param_value           IN  dpp_parameters.prr_value%TYPE        := NULL
 , p_descr_eng             IN  dpp_parameters.descr_eng%TYPE        := NULL
 , p_descr_fra             IN  dpp_parameters.descr_fra%TYPE        := NULL
 , p_instance_name         IN  dpp_parameters.ite_name%TYPE         := NULL
 , p_date_creat            IN  dpp_parameters.date_creat%TYPE       := NULL
 , p_user_creat            IN  dpp_parameters.user_creat%TYPE       := NULL
 , p_date_modif            IN  dpp_parameters.date_modif%TYPE       := NULL
 , p_user_modif            IN  dpp_parameters.user_modif%TYPE       := NULL
);
```

Parameters:

- **p_param_name:** parameter name
- **p_param_value:** parameter value
- **p_descr_eng:** description in English
- **p_descr_fra:** description in French
- **p_instance_name:** database instance name
- **p_date_creat:** creation date

- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.32. insert_recipient

This procedure inserts a new email recipient in the **dpp_recipients** table. The schema can be identified by its ID or by its functional name which is unique.

Format:

```
PROCEDURE insert_recipient(
   p_schema_id             IN  dpp_recipients.sma_id%TYPE            := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE      := NULL
 , p_email_addr            IN  dpp_recipients.email_addr%TYPE
 , p_date_creat            IN  dpp_recipients.date_creat%TYPE        := NULL
 , p_user_creat            IN  dpp_recipients.user_creat%TYPE        := NULL
 , p_date_modif            IN  dpp_recipients.date_modif%TYPE        := NULL
 , p_user_modif            IN  dpp_recipients.user_modif%TYPE        := NULL
);
```

Parameters:

- **p_schema_id:** ID of the schema
- **p_functional_name:** functional name of the schema
- **p_email_addr:** email address
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.33. delete_recipient

This procedure deletes an email recipient in the **dpp_recipients** table. The schema can be identified by its ID or by its functional name which is unique.

Format:

```
PROCEDURE delete_recipient(
   p_schema_id             IN  dpp_recipients.sma_id%TYPE            := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE      := NULL
 , p_email_addr            IN  dpp_recipients.email_addr%TYPE
);
```

Parameters:

- **p_schema_id:** ID of the schema
- **p_functional_name:** functional name of the schema
- **p_email_addr:** email address

6.2.14.34. duplicate_recipient

This procedure duplicates an email recipient in the **dpp_recipients** table from a schema to another one. The schemas can be identified by their ID or by their functional name which is unique.

Format:

```
PROCEDURE duplicate_recipient(
   p_schema_id             IN  dpp_recipients.sma_id%TYPE         := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE   := NULL
 , p_email_addr            IN  dpp_recipients.email_addr%TYPE
 , p_trg_schema_id         IN  dpp_recipients.sma_id%TYPE         := NULL
 , p_trg_functional_name   IN  dpp_schemas.functional_name%TYPE   := NULL
 , p_date_creat            IN  dpp_recipients.date_creat%TYPE     := NULL
 , p_user_creat            IN  dpp_recipients.user_creat%TYPE     := NULL
 , p_date_modif            IN  dpp_recipients.date_modif%TYPE     := NULL
 , p_user_modif            IN  dpp_recipients.user_modif%TYPE     := NULL
);
```

Parameters:

- **p_schema_id:** ID of the source schema
- **p_functional_name:** functional name of the source schema
- **p_email_addr:** email address
- **p_trg_schema_id:** ID of the target schema
- **p_trg_functional_name:** functional name of the target schema
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.35. insert_schema_option

This procedure inserts a new schema option in the **dpp_schema_options** table. The schema can be identified by its ID or by its functional name which is unique.

Format:

```
PROCEDURE insert_schema_option(
   p_schema_id             IN  dpp_schema_options.sma_id%TYPE        := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE      := NULL
 , p_option_name           IN  dpp_schema_options.otn_name%TYPE
 , p_option_value          IN  dpp_schema_options.stn_value%TYPE
 , p_usage                 IN  dpp_schema_options.stn_usage%TYPE
 , p_date_creat            IN  dpp_schema_options.date_creat%TYPE    := NULL
 , p_user_creat            IN  dpp_schema_options.user_creat%TYPE    := NULL
 , p_date_modif            IN  dpp_schema_options.date_modif%TYPE    := NULL
 , p_user_modif            IN  dpp_schema_options.user_modif%TYPE    := NULL
);
```

Parameters:

- **p_schema_id:** ID of the schema
- **p_functional_name:** functional name of the schema
- **p_option_name:** option name

- **p_option_value:** option value
- **p_usage:** usage
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.36. delete_schema_option

This procedure deletes a schema option from the **dpp_schema_options** table. The schema can be identified by its ID or by its functional name which is unique.

Format:

```
PROCEDURE delete_schema_option(
   p_schema_id             IN  dpp_schema_options.sma_id%TYPE        := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE      := NULL
 , p_option_name           IN  dpp_schema_options.otn_name%TYPE
 , p_usage                 IN  dpp_schema_options.stn_usage%TYPE
 , p_option_value          IN  dpp_schema_options.stn_value%TYPE
);
```

Parameters:

- **p_schema_id:** ID of the schema
- **p_functional_name:** functional name of the schema
- **p_option_name:** option name
- **p_usage:** usage
- **p_option_value:** option value

6.2.14.37. update_schema_option

This procedure updates a schema option in the **dpp_schema_options** table. The schema can be identified by its ID or by its functional name which is unique.

Format:

```
PROCEDURE update_schema_option(
   p_schema_id             IN  dpp_schema_options.sma_id%TYPE        := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE      := NULL
 , p_option_name           IN  dpp_schema_options.otn_name%TYPE
 , p_option_value          IN  dpp_schema_options.stn_value%TYPE
 , p_usage                 IN  dpp_schema_options.stn_usage%TYPE
 , p_option_new_value      IN  dpp_schema_options.stn_value%TYPE     := NULL
 , p_date_creat            IN  dpp_schema_options.date_creat%TYPE    := NULL
 , p_user_creat            IN  dpp_schema_options.user_creat%TYPE    := NULL
 , p_date_modif            IN  dpp_schema_options.date_modif%TYPE    := NULL
 , p_user_modif            IN  dpp_schema_options.user_modif%TYPE    := NULL
);
```

Parameters:

- **p_schema_id:** ID of the schema
- **p_functional_name:** functional name of the schema
- **p_option_name:** option name
- **p_option_value:** current option value
- **p_usage:** usage
- **p_option_new_value**: new option value
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.38. duplicate_schema_option

This procedure duplicates a schema option in the **dpp_schema_options** table from a schema to another one. The schemas can be identified by their ID or by their functional name which is unique.

Format:

```
PROCEDURE duplicate_schema_option(
   p_schema_id             IN  dpp_schema_options.sma_id%TYPE        := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE      := NULL
 , p_option_name           IN  dpp_schema_options.otn_name%TYPE
 , p_usage                 IN  dpp_schema_options.stn_usage%TYPE
 , p_option_value          IN  dpp_schema_options.stn_value%TYPE
 , p_trg_schema_id         IN  dpp_schema_options.sma_id%TYPE        := NULL
 , p_trg_functional_name   IN  dpp_schemas.functional_name%TYPE      := NULL
 , p_date_creat            IN  dpp_schema_options.date_creat%TYPE    := NULL
 , p_user_creat            IN  dpp_schema_options.user_creat%TYPE    := NULL
 , p_date_modif            IN  dpp_schema_options.date_modif%TYPE    := NULL
 , p_user_modif            IN  dpp_schema_options.user_modif%TYPE    := NULL
);
```

Parameters:

- **p_schema_id:** ID of the source schema
- **p_functional_name:** functional name of the source schema
- **p_option_name:** option name
- **p_usage:** usage
- **p_option_vallue:** option value
- **p_trg_schema_id:** ID of the target schema
- **p_trg_functional_name:** functional name of the target schema
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.39. insert_schema_relation

This procedure inserts a relationship between two schemas in the **dpp_schema_relations** table. The source and target schemas can be identified by their ID or functional name which is unique.

Format:

```
PROCEDURE insert_schema_relation(
   p_schema_id_from        IN  dpp_schema_relations.sma_id_from%TYPE := NULL
 , p_functional_name_from  IN  dpp_schemas.functional_name%TYPE      := NULL
 , p_schema_id_to          IN  dpp_schema_relations.sma_id_to%TYPE   := NULL
 , p_functional_name_to    IN  dpp_schemas.functional_name%TYPE      := NULL
 , p_date_from             IN  dpp_schema_relations.date_from%TYPE   := NULL
 , p_date_to               IN  dpp_schema_relations.date_to%TYPE     := NULL
 , p_date_creat            IN  dpp_schema_relations.date_creat%TYPE  := NULL
 , p_user_creat            IN  dpp_schema_relations.user_creat%TYPE  := NULL
 , p_date_modif            IN  dpp_schema_relations.date_modif%TYPE  := NULL
 , p_user_modif            IN  dpp_schema_relations.user_modif%TYPE  := NULL
);
```

Parameters:

- **p_schema_id_from:** ID of the source schema
- **p_functional_name_from:** functional name of the source schema
- **p_schema_id_to:** ID of the target schema
- **p_functional_name_to:** functional name of the target schema
- **p_date_from:** validity start date
- **p_date_to:** validity end date
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.40. delete_schema_relation

This procedure deletes a relationship between two schemas from the **dpp_schema_relations** table. The source and target schemas can be identified by their ID or functional name which is unique.

Format:

```
PROCEDURE delete_schema_relation(
   p_schema_id_from        IN  dpp_schema_relations.sma_id_from%TYPE := NULL
 , p_functional_name_from  IN  dpp_schemas.functional_name%TYPE      := NULL
 , p_schema_id_to          IN  dpp_schema_relations.sma_id_to%TYPE   := NULL
 , p_functional_name_to    IN  dpp_schemas.functional_name%TYPE      := NULL
);
```

Parameters:

- **p_schema_id_from:** ID of the source schema
- **p_functional_name_from:** functional name of the source schema
- **p_schema_id_to:** ID of the target schema
- **p_functional_name_to:** functional name of the target schema

6.2.14.41. update_schema_relation

This procedure updates a relationship between two schemas in the **dpp_schema_relations** table. The source and target schemas can be identified by their ID or functional name which is unique.

Format:

```
PROCEDURE update_schema_relation(
   p_schema_id_from        IN  dpp_schema_relations.sma_id_from%TYPE := NULL
 , p_functional_name_from  IN  dpp_schemas.functional_name%TYPE      := NULL
 , p_schema_id_to          IN  dpp_schema_relations.sma_id_to%TYPE   := NULL
 , p_functional_name_to    IN  dpp_schemas.functional_name%TYPE      := NULL
 , p_date_from             IN  dpp_schema_relations.date_from%TYPE   := NULL
 , p_date_to               IN  dpp_schema_relations.date_to%TYPE     := NULL
 , p_date_creat            IN  dpp_schema_relations.date_creat%TYPE  := NULL
 , p_user_creat            IN  dpp_schema_relations.user_creat%TYPE  := NULL
 , p_date_modif            IN  dpp_schema_relations.date_modif%TYPE  := NULL
 , p_user_modif            IN  dpp_schema_relations.user_modif%TYPE  := NULL
);
```

Parameters:

- **p_schema_id_from:** ID of the source schema
- **p_functional_name_from:** functional name of the source schema
- **p_schema_id_to:** ID of the target schema
- **p_functional_name_to:** functional name of the target schema
- **p_date_from:** validity start date
- **p_date_to:** validity end date
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

6.2.14.42. clean_up_job_runs

This procedure cleans up the job runs (**dpp_job_runs** table and **dpp_job_logs** table) belonging to database schema. The schema can be identified by its ID or by its functional name which is unique.

Format:

```
PROCEDURE clean_up_job_runs(
   p_schema_id             IN  dpp_job_runs.sma_id%TYPE           := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE   := NULL
 , p_date_started          IN  dpp_job_runs.date_started%TYPE     := NULL
 , p_date_ended            IN  dpp_job_runs.date_ended%TYPE       := NULL
);
```

Parameters:

- **p_schema_id:** ID of the schema whose logs must be cleaned up
- **p_functional_name:** functional name of the schema whose logs must be cleaned up
- **p_date_started:** start date of the period whose the logs must be cleaned up
- **p_date_ended:** end date of the period whose the logs must be cleaned up

6.2.14.43. duplicate_schema_config

This function duplicates a schema and all configurations attached to it (**dpp_nodrop_objects**, **dpp_actions**, **dpp_recipients** and **dpp_schema_options**). The source schema can be identified by its ID or by its functional name which is unique. If the target schema ID is not passed as parameter, it is computed by the function.

Format:

```
FUNCTION duplicate_schema_config(
   p_src_schema_id         IN  dpp_schemas.sma_id%TYPE            := NULL
 , p_src_functional_name   IN  dpp_schemas.functional_name%TYPE   := NULL
 , p_trg_schema_id         IN  dpp_schemas.sma_id%TYPE            := NULL
 , p_trg_functional_name   IN  dpp_schemas.functional_name%TYPE
 , p_date_creat            IN  dpp_schemas.date_creat%TYPE        := NULL
 , p_user_creat            IN  dpp_schemas.user_creat%TYPE        := NULL
 , p_date_modif            IN  dpp_schemas.date_modif%TYPE        := NULL
 , p_user_modif            IN  dpp_schemas.user_modif%TYPE        := NULL
) RETURN dpp_schemas.sma_id%TYPE;
```

Parameters:

- **p_src_schema_id:** ID of the source schema
- **p_src_functional_name:** functional name of the source schema
- **p_trg_schema_id:** ID of the target schema
- **p_trg_functional_name:** functional name of the target schema
- **p_date_creat:** creation date
- **p_user_creat:** creation user ID
- **p_date_modif:** last modification date
- **p_user_modif:** last modification user ID

Return: the target schema ID

6.2.14.44. delete_schema_config

This procedure deletes a schema and all configurations attached to it (**dpp_nodrop_objects**, **dpp_actions**, **dpp_recipients** and **dpp_schema_options**). The schema can be identified by its ID or by its functional name which is unique.

Format:

```
PROCEDURE delete_schema_config(
   p_schema_id             IN  dpp_schemas.sma_id%TYPE            := NULL
 , p_functional_name       IN  dpp_schemas.functional_name%TYPE   := NULL
);
```

Parameters:

- **p_schema_id:** ID of the schema
- **p_functional_name:** functional name of the schema

6.2.14.45. Error codes

The procedures and functions of the **dpp_cnf_krn** package can raise several exceptions when some errors or problems are detected. Those are the application error codes that can be raised and the corresponding error description.

Application error code	Description
-20001	Invalid parameter passed to a procedure or function.
-20002	The current user does not have the needed privileges to create a procedure in another schema.
-20003	The current user does not have the needed privileges to execute a procedure in another schema.
-20004	The current user does not have the needed privileges to drop a procedure in another schema.
-20005	The instance name already exists.
-20006	The instance does not exist.
-20007	The instance is referenced by some child data.
-20008	The schema type name already exists.
-20009	The schema type does not exist.
-20010	The schema type is referenced by some child data.
-20011	The role name already exists.
-20012	The role does not exist.
-20013	The role is referenced by some child data.
-20014	The schema already exists.
-20015	The schema functional name already exists.
-20016	The schema functional name does no exist.
-20017	The schema does not exist.
-20018	The schema is referenced by some child data.
-20019	The "no drop" object already exists.
-20020	The "no drop" object does no exist.
-20021	The action name already exists.
-20022	The action does not exist.
-20023	The parameter already exists.
-20024	The parameter does not exist.
-20025	The email recipient already exists.
-20026	The email recipient does not exist.
-20027	The option name does not exist.
-20028	The option already exists.

Application error code	Description
-20029	The option does not exist.
-20030	The source and target schemas are the same.
-20031	The relationship between schemas already exists.
-20032	The relationship between schemas does not exist.

6.2.15. DPP_CNF_VAR package

This package contains global variables and constants for the **dpp_cnf_krn** package, to make this one stateless.

6.2.16. DC_DBA_MGMT_KILL_SESS_DEDIC_DB stored procedure

This stored procedure kills a session.

Caution: This stored procedure is only deployed in a COP installation.

Format:

```
PROCEDURE dc_dba_mgmt_kill_sess_dedic_db(
   session_sid      IN INT
 , session_serial   IN INT
 ,v_inst_id         IN NUMBER DEFAULT 0
);
```

Parameters:

- **session_id:** the SID of the session to be killed
- **session_serial:** The serial number of the session to be killed
- **v_inst_id:** The instance identifier

6.2.17. DC_DBA_MGMT_LOCK_USER stored procedure

This procedure locks or unlocks a schema.

Caution: This stored procedure is only deployed in an AWS or COP installation.

Format:

```
PROCEDURE dc_dba_mgmt_lock_user(
   user             IN VARCHAR2
 , b_lock           IN BOOLEAN
);
```

Parameters:

- **user:** name of the schema to be locked or unlocked
- **b_lock: TRUE** if the schema needs to locked, **FALSE** if it needs to be unlocked.

6.2.18. DPP_MONITORING job

This job is part of the implementation of the timeout monitoring. Every five minutes, it executes the **exec_monitoring** procedure of the **dpp_monitoring_krn** package.