EC PL/SL Toolkit v24.9 - Technical Guide

Author: Philippe Debois (European Commission - DIGIT)

Table of Contents

1. Introduction

1.1. Audience

This guide is intended for Oracle database developers of the EC responsible for the development, maintenance, and packaging of the EC PL/SQL toolkit. It explains its content and structure, how to incorporate new versions of the tools, and how to package them. By providing detailed procedures, this guide aims to assist EC developers in successfully releasing new versions of the toolkit.

2. Toolkit home directory

The home directory of the toolkit (`ec_plsql_toolkit`) contains the following folders:

Name	Long name	Usage
`apps`	Applications dir	Contains the material of all tools included in the toolkit.
`bin`	Binaries dir	Contains Windows and Linux shell scripts of the installer.
`conf`	Configuration dir	Contains configuration files of the installer.
`doc`	Documentation dir	Contains the documentation of the toolkit.
`sql`	SQL dir	Contains the latest version of the SQL scripts of the installer.
`tmp`	Temporary dir	Contains temporary files (cleaned up automatically upon startup).

Name	Long name	Usage
`logs`	Logs dir	Contains log files (must be cleaned up manually).

You will note that all directories except `apps` and `doc` are related to the installer of the toolkit (the DBM or **D**ata**B**ase **M**igration tool).

The home directory also contains the following files:

File	Usage
clean-up	Shell script to clean-up temporary and logs files before making a zip.
contributing-guidelines.md	Guidelines for contributing to this project.
dbm-cli	Main shell script (DBM client).
dbm-cli.sql	SQL scripts to execute DBM commands.
ec-cla.pdf	Contributor licence agreement.
LICENCE.txt	Licence terms.
migrate-dbm	Shell script to migrate the DBM tool.
mkzip	Shell script to package the toolkit.
NOTICE.txt	Licence terms of embedded 3rd party components.
README.md	Readme file.
set-os	Shell script to determine current operating system.

2.1. Apps directory

The `apps` folder contains one subfolder for each tool and a `00_all` subfolder for material common to all tools. These sub-folders are prefixed with a sequence number that reflects the order in which the tools must be installed, taking into account their dependencies.

An alternate location for the `apps` folder can be specified by setting the `DBM_APPS_DIR` environment variable but this possibility is not used for the toolkit.

The file structure is the following:

```
apps
├── 00_all
│   └── releases
│       ├── xx.yy[.zz] # toolkit version
│       └── all
│           ├── conceal
│           ├── config
│           ├── expose
│           └── uninstall
└── <xx_tool>
    ├── doc
    ├── demo
    ├── tuto
    └── releases
        ├── xx.yy[.zz] # tool version
        │   ├── config
        │   ├── install
        │   │   └── rollback
        │   └── upgrade
        │       └── rollback
        └── all
            ├── config
            ├── precheck
            ├── setup
            └── uninstall
```

Each `releases` folder contains an `all` version that holds the material that do not depend on a specific version (i.e., that is common to all versions). This folder also contains a `Release-Notes.txt` file that documents new features, changes (enhancements), and bug fixes of each release.

Each version folder (`all` or `xx.yy[.zz]`) contains subfolders whose name is dictated by the corresponding installer command plus a `config` folder containing configuration files.

2.2. Bin directory

This directory contains shell scripts that are used by the installer for performing some specific tasks. Two versions of each file are provided, one for Windows (`.bat` or `.cmd` extension), one for Linux (no extension but with execute permission).

Here follows the list of provided files and their usage (only the Linux files are listed):

File	Usage
delete-file	Delete a file whose path is passed in parameter.

File	Usage
get-hashes	Compute the hash of all files located in the directory passed in parameter.
plus	Launch SQL*Plus and execute the script passed in parameter.
plus-loop	Not used.
read-file	Read the content of a file passed in parameter.
read-files	Read the content of all files in the directory passed in parameter.
scan-files	Scan the file system under the directory passed in parameter.
sqldba	Launch SQL*Plus (connecting as DBA) and execute the script passed in parameter.

2.3. Conf directory

This directory contains the configuration files of the DBM installer.

File	Usage
dbm_utility.conf	Configuration file of the DBM installer.

The `DBM_CONF_PATH` environment variable can be set to store the file in another location.

2.4. Doc directory

This directory contains the documentation of the toolkit in several formats (MarkDown, PDF, HTML).

File	Usage
ec_plsql_toolkit_installation_guide	Installation guide of the toolkit.
ec_plsql_toolkit_technical_guide	Technical guide of the toolkit.

2.5. Logs directory

This directory contains the logs of all installer commands that have been executed. It is the responsibility of end-users to clean-up this directory (i.e., log files are never deleted automatically).

An alternate location can be specified by setting the `DBM_LOGS_DIR` environment variable.

2.6. Sql directory

This folder contains - the latest version of - all SQL scripts of the DBM installer.

File	Usage
check-dbm.sql	Check whether the DBM tool is correctly installed.
dbm_uninstall.sql	Uninstall all database objects of the DBM tool
dbm-checksums.sql	Compute the global checksum of database objects and packages.
dbm-checksums-detailed.sql	Compute the checksum of each database object.
dbm-startup.sql	Executed at startup of the `dbm-cli` .
migrate-dbm.sql	Perform installation and/or upgrade of the DBM tool.

2.7. Tmp directory

This directory contains the temporary files created by the installer during its operations. It is automatically cleaned up when the `dbm-cli` shell script is executed. All temporary files are prefixed with a tilde (`~`) and are created in this folder, except for the `~set-os.sql` file, which is created in the home directory. The latter file is used to set the operating system and also to detect the first run of `dbm-cli` after an unzip.

An alternate location can be specified by setting the `DBM_TMP_DIR` environment variable.

3. Tools migration setup

This section describes how the migration of the tools is configured.

3.1. Version specific setup

Installation scripts, upgrade scripts, and database object inventory are specific to each version and placed respectively in the `install` , `upgrade` , and `config` subfolders of the `xx.yy[.zz]` version folder.

It is recommanded to use changelog files (`install.dbm` and `upgrade.dbm`) to tell the installer which script to execute to install or upgrade a tool. This method allows conditional file execution, offers better error recovery, and support rollback in case of migration failure. Master scripts (`install.sql` and `upgrade.sql`) inherited from the former pl/sql installer are still in use for some tools and should ideally be converted when a new version is released.

For now, no `rollback` subfolder exists, and no rollback scripts are provided. These may be developed in the future if deemed valuable.

3.2. Version independent setup

Uninstallation scripts, precheck scripts, setup scripts, and some parameters common to all versions are placed respectively in the `install`, `setup`, `precheck`, and `config` subfolder of the `all` version folder.

3.2.1. Scripts

Uninstallation scripts drop all database objects and revoke any granted access right. Setup scripts are used by the QC tool only. Precheck scripts are executed before any migration to check prerequisites.

3.2.2. Parameters

The configuration file `config/xxx_utility.conf` can contain the following parameters:

Name	Usage
xxx_utility.par.app_alias	Application or tool alias.
xxx_utility.par.object_name_pattern (*)	Pattern defining which DB objects belong to the tool.
xxx_utility.par.object_name_anti_pattern	Pattern to exclude some DB objects.
xxx_utility.par.expose_pattern	Pattern defining which DB objects must be exposed to other schemas.
xxx_utility.par.expose_anti_pattern	Pattern to exclude exposure of some DB objects.
xxx_utility.par.expose_read_only_pattern	Pattern defining which DB objects must be exposed in read-only mode to other schemas.
xxx_utility.par.requires	list of tools it depends on

(*) mandatory parameter i.e., it must be defined for any tool.

All patterns are regular expressions that are checked against object types and names via the SQL expression `REGEXP_LIKE(object_type||' '||object_name, '<pattern>')`. To check that objects do not match anti-patterns, this expression is prefixed with a `NOT`.

The `requires` parameter is a comma separated list of tools (plus an optional versions) that are required by the current tool and that must be installed beforehand. Tools having a sequence number 20 or above depend on others and should have this parameter defined.

3.2.3. Substitution variables

The configuration file `config/xxx_utility.conf` can also contain the definition of some substitution variables whose value is supplied by end-user interactively

Here follows the parameters that can be defined for each substitution variable:

Name	Usage
xxx_utility.var.<name>.name	Variable name.
xxx_utility.var.<name>.descr	Variable description (used when prompting end-user).
xxx_utility.var.<name>.seq	Sequence in which variable value is prompted.
xxx_utility.var.<name>.data_type	Data type (CHAR or NUMBER).
xxx_utility.var.<name>.nullable	Y=optional, N=mandatory.
xxx_utility.var.<name>.convert_value_sql	SQL expression to convert the entered value.
xxx_utility.var.<name>.default_value_sql	SQL expression to provide a default value.
xxx_utility.var.<name>.check_value_sql	SQL expression to check entered value.
xxx_utility.var.<name>.check_error_msg	Message displayed in case of check failure.

The value of a substitution variable can also be passed in command line with the following syntax: `dbm-cli "-xxx_utility.var.<name>.value=<value>".

3.2.4. Privileges

For tools that require specific system privileges, roles, or grants for their installation and/or operation, a `privileges.dbm` file is also created in the `config` subfolder.

3.3. Common setup

Setup files common to (all versions of) all tools are stored under the `apps/00_all/releases/all` folder.

3.3.1. Common scripts

Scripts that implement expose, conceal, and uninstall commands and that are common to all tools are stored in the `expose`, `conceal`, and `uninstall` subfolders.

Files for the expose command are the following:

Name	Usage
check_grantee.sql	Check that grantee is `PUBLIC` or correspond to an existing schema
check_sys_privs.sql	Check that current schema has the necessary privileges to create any or public synonyms (depending on the grantee).
create_synonyms.sql	Create private or public synonyms (depending on the grantee).
expose.dbm	changelog file listing scripts to execute.
grant_access.sql	Grant access to database objects that must be exposed.

Files for the conceal command are the following:

Name	Usage
check_grantee.sql	Check that grantee is `PUBLIC` or correspond to an existing schema
check_sys_privs.sql	Check that current schema has the necessary privileges to drop any or public synonyms (depending on the grantee).
conceal.dbm	changelog file listing scripts to execute.
drop_synonyms.sql	Drop private or public synonyms (depending on the grantee).
revoke_access.sql	Revoke access to previously exposed database objects.

For for the uninstall command are the following:

Name	Usage
drop_all_synonyms.sql	Drop all synonyms on exposed database objects.
revoke_all_access.sql	Revoke all grants on exposed database objects.
uninstall.dbm	changelog file listing scripts to execute.

3.3.2. Common privileges

Privileges that are required to install or operate any tool of the toolkit is defined in the `privileges.dbm` file. Usually, for installation, the schema must have the system privileges to open a session, query data dictionary views (catalog), and create usual database objects (tables, views, sequences, types, packages).

3.3.3. Common substitution variables

Substitution variables that are common to all tools are defined in the `all.conf` configuration file stored in the `config` subfolder.

The following common substitution variables are defined:

Name	Usage	Allowed values
env	Environment (Dev, Test, Acceptance, Stress test, Production), derived from schema name.	D, T, A, S, P
installation_env	Installation environment (Data Cener, Cloud on Prem, Amazon Cloud, Others).	DC, COP, AWS, OTH
tab_ts	Tablespace in which tables will be created.	Any valid tablespace name
idx_ts	Tablespace in which indexes will be created.	Any valid tablespace name
ut_plsql	Is the utPLSQL framework (PL/SQL unit testing) installed? Used to conditionally install unit testing packages.	Y, N

Installation environment (AWS=Amazon RDS, COP=Cloud on Premises in EC Data Center, DC=Legacy DB in EC Data Centre, or OTH=Other (e.g., any non-EC DC)) is determined automatically based on a query and should in principle not be modified.

Tablespaces are by default those that are the most often used by tables and indexes in the current schema. When the schema does not hold any table already, value must be defined by end-user via the `configure` interactive command, or by passing their value in parameter of the `dbm-cli` shell script.

4. Procedures

This section describes how to incorporate a new version of a tool into the toolkit and how to package a new version of the toolkit. It also explains the specific steps and additional considerations required to prepare a new version of the DBM tool.

4.1. Creating a new version of a tool

The steps are the following:

- Prepare a new version of a tool.
- Define a release strategy and assign a release number.
- Create a release folder and its subfolders.
- Populate `install` and/or `upgrade` subfolders.

- Make database objects inventory including checksums.
- Clean up full release folders
- Test the migration
- Update release notes (while making changes).
- Make a new version of the toolkit.

4.1.1. Preparing a new version of a tool

Each tool has a home directory, located outside the toolkit's main directory, that contains the latest version or work-in-progress of its materials (source code, database scripts, documentation, data sets, etc.). This is where the new version must be prepared.

Database objects fall into one of the following two categories: database structure (tables, views, constraints, indexes, sequences, etc.) and database code (packages, procedures, functions, etc.).

For database structure, separate SQL scripts should be prepared to support both full installation (from scratch) and incremental upgrades. The master script for full installation is commonly named `xxx_objects.sql` (where `xxx` is the abbreviation of the tool). These SQL scripts are typically stored in a `sql` subfolder within the tool's home directory.

For tools that utilise the conditional statement execution and/or statement-level recovery features, the master script should be named xxx_objects.dbm.sql. The .dbm.sql double extension is used to activate these features. Note that the size of this file must be kept below the limit of 32k and should be split into multiple files if necessary.

For database code, a single file can be used for both types of migration, provided the `CREATE OR REPLACE` statement is used to create or replace the database object. These PL/SQL scripts should bear the same name as their corresponding database object, with an extension that reflects their type (e.g., `.pks` for package specification, `.pkb` for package body, `.prc` for procedures, `.fnc` for functions, etc.). Store these PL/SQL scripts in a `plsql` subfolder within the tool's home directory.

To facilitate rollback or recovery in case of migration failure, consider the following recommendations:

- Separate DDL and DML operations: Place DDL operations (Data Definition Language) in separate files from DML operations (Data Manipulation Language). This practice helps in isolating schema changes from data changes, making it easier to manage and recover from failures.
- Use multiple files based on object types: Instead of combining all operations into one large script, organise scripts by object types (e.g., tables, views, constraints, indexes, sequences). This approach enhances clarity and allows for targeted execution and recovery.
- Tag your SQL statements with an identifier (`REM DBM-XXXXX`) to allow statement level recovery or condition statement execution.

- Ensure re-entrant scripts whenever possible: Make scripts re-entrant so that they can be executed multiple times without causing errors or unintended changes. This capability ensures robustness and reliability during deployment and rollback scenarios.
- Provide rollback scripts whenever possible: Include rollback scripts alongside deployment scripts. Rollback scripts should revert changes made by the deployment scripts, enabling quick and controlled rollback procedures in case of issues.

4.1.2. Defining a release strategy and assigning a release number

A version of a tool can be released to address bugs, implement minor enhancements, introduce new features, or combine any of these objectives.

Releases are numbered as `xx.yy[.zz]`, where:

- `xx` denotes the major release number, always corresponding to the year of the release (e.g., 24 for 2024), aligning with Oracle's release numbering.
- `yy` represents the minor release number, a sequential number within the year starting with 0.
- `zz` indicates the bugfix release number and is optional, a sequential number within the minor release starting with 1.

Depending on the significance and nature (structure and/or code) of the changes, you may choose to provide installation scripts, upgrade scripts, or both. Here are the recommended practices:

- Installation scripts only (very rare): Used for the very first version of a tool or when there are significant changes to the data model that do not allow data migration from a previous version.
- Upgrade scripts only: Applied when only database code is modified (e.g., packages), without any structural changes. This practice can be used for bug fixes and/or minor changes.
- Both types of scripts: Recommended for all other scenarios where both structural changes and code modifications are implemented.

It's important to note that the Database Management (DBM) tool will determine which scripts to apply based on the content of the target schema.

4.1.3. Creating a release folder and its subfolders

Once the release strategy and version number have been defined, the release materials can be integrated into the toolkit.

Create a release folder within `apps/xx_<tool>/releases` named after the defined version number. Within this folder, include the following subfolders based on the defined release strategy:

- `config` : For storing configuration files.
- `install` : For housing installation scripts (include a `rollback` subfolder if applicable).
- `upgrade` : For storing upgrade scripts (include a `rollback` subfolder if applicable).

Alternatively, you can duplicate a previous release folder to ensure that essential files, especially those in the `config` and `install` subfolders, are not overlooked. This method helps maintain consistency and completeness across releases.

4.1.4. Populate install and/or upgrade subfolders

Copy scripts responsible for full installation in the `install` subfolder and those responsible for upgrade in the `upgrade` subfolder. Remember, files containing database code are identical for both installation and upgrade, on the contrary to files related to the database structure.

To provide better support for rollback in case of migration failure, it is recommended to use the changelogs method of the DBM tool, i.e. to create `install.dbm` and `upgrade.dbm` changelog files. This method also allows the conditional execution of files depending on the target environment.

4.1.5. Making database objects inventory including checksums

The inventory of database objects (stored in the file `config/objects.dbm`), which includes their checksums, enables validation of the correct installation or upgrade of the tool. It also provides a means to determine the currently installed version of the tool if this information is lost (e.g., due to re-installation of the DBM tool).

To generate the database objects inventory, execute the `dbm-cli` shell script while connected to the schema containing the database objects being released. Use the following command: `@dbm-cli make <tool> <version>` . Here, `<tool>` refers to the name or abbreviation of the tool, and `<version>` is the designated target release number. Although the version parameter is optional, it is recommended to explicitly specify it, as the currently known installed version managed by the DBM tool may not be up to date.

4.1.6. Cleaning up full release folders

The size of the toolkit bundle can grow significantly if all versions of all tools are kept indefinitely. At a minimum, the toolkit should maintain the latest full release and all incremental releases, ensuring users with older versions can upgrade to the latest version seamlessly. Preserving an inventory of all releases is crucial for the DBM tool to accurately identify the currently installed version of each tool and determine necessary upgrades.

On the other hand, it is useful to be able to reinstall a previous version of the toolkit, for instance, to reproduce a bug reported by an end-user, to test the DBM tool itself, or to revert to a more stable version. Maintaining a history of all toolkit bundles (archives) achieves thi goal.

It is recommended to:

- Keep 2 full releases (i.e., the `install` folder for 2 releases).
- Keep all incremental releases (i.e., all `upgrade` folders).
- Keep all inventories and other configuration files (i.e., all `config` folders).

`install` folders that are removed from the toolkit bundle must be archived in the `ec_plsql_toolkit_archive` folder using the `git mv` command so as to be able to restore them when necessary. This archive folder as the same structure as the `ec_plsql_toolkit` folder.

For vulnerability assessment and penetration testing, a single version of each file should be provided. The simplest approach is to retain only the `install` folder of the latest full release and apply any changes from subsequent incremental releases if necessary.

4.1.7. Testing the migration

Migration should be rigorously tested on a schema that does not already contain the version of the tool being released. Both full installation and incremental upgrades should undergo testing. Testing the full installation of a tool requires its prior uninstallation for accurate evaluation.

4.1.8. Updating release notes

Update the `Release-Notes.txt` file located in the `releases` folder. This file is cumulative and thereby contains the history of all versions since the creation of the tool.

Add a new section for the release formatted as the following:

```
VERSION xx.yy[.zz] - Month YYYY

New features:
- ...

Changes:
- ...

Bug fixing:
- ...
```

Put `- n/a` when a section is not applicable.

Adapt the file header with the correct release number and date:

```
XXX Utility xx.yy[.zz] Release Notes
Date: Month YYYY
```

To ensure a correct and complete documentation of the release, it is recommended to update this file while the software is being adapted.

4.2. Creating a new version of the installer

The procedure for creating a new version of the DBM installer is similar to that for other tools but includes some minor differences and additional steps.

The DBM installer can upgrade itself only if its session is not invalidated by one of the database changes. For instance, altering a table referenced in the `DBM_UTILITY_VAR` package will invalidate it, raising the "ERROR-04068: Existing state of packages has been discarded" exception the next time one of the DBM packages is invoked.

To circumvent this problem, a migration solution specific to the DBM installer has been developed. This solution takes the form of a SQL script (`migrate-dbm.sql`) executed with SQL*Plus from a shell script (`migrate-dbm[.cmd]`).

This ad-hoc solution can determine which version of the DBM tool is currently installed (thanks to checksums) and apply successive upgrades until the latest available version (or up to the one passed as a parameter to the shell script).

To achieve this goal, the `migrate-dbm.sql` script must be adapted for each new release. The version number and upgrade type (full installation and/or incremental upgrade) must be declared. Checksums on database structure and database packages must also be computed and defined in the SQL script itself.

4.2.1. Declaring release number and type

Each new release of the DBM tool must be declared in the `initially()` procedure of the `migrate-dbm.sql` script as follows.

If the release includes full installation scripts (i.e., it contains an `install` subfolder under the version folder), add the following line:

```
t_ins(<ver-nbr>) := '<ver-code>';
```

Where `<ver-nbr>` is the version number and `<ver-code>` is the version code. The version code is formatted as `xx.yy[.zz]` (the bugfix release number being optional). The version number can be computed as `xx * 10000 + yy * 100 + zz` .

If a release includes upgrade scripts (i.e., it contains an `upgrade` subfolder under the version folder), add the following line:

```
-- Initialise versions
t_upg(<ver-nbr>) := '<ver-code>';
```

Here is an example of how versions 24.0, 24.1, and 24.3 have been declared:

```
-- Initialise versions
t_ins(240000) := '24.0'; -- installable only
t_ins(240100) := '24.1';   t_upg(240100) := '24.1'; -- installable and upgradable
                           t_upg(240300) := '24.3'; -- upgradable only
```

If you copy/paste code, make sure to change and align version number and version code accordingly!

4.2.2. Declaring checksums

Two different checksums are computed: one for structural database objects (tables, indexes, constraints, sequences, etc.) and one for procedural database objects (packages, procedures, functions, etc.).

To compute these two checksums, execute the query contained in the `sql/dbm-checksums` file while connected to a schema where the database objects of the release have already been deployed.

Add the following lines of code in the `initially()` procedure of the `sql/migrate-dbm.sql` scripts:

```
-- Initialise checksums for objects and packages
t_obj_sum(<ver-nbr>) := <obj-checksum>;
t_pkg_sum(<ver-nbr>) := pkg-checksum;
```

where `<obj-checksum>` is the checksum of the structure and `<pkg-checksum>` the checksum of the code.

Here follows an example of checksums declaration for the two first versions:

```
-- Initialise checksums for objects and packages
t_obj_sum(240000) := 671291450; t_pkg_sum(240000) := 296702143;
t_obj_sum(240100) := 544776930; t_pkg_sum(240100) := 1062429868;
```

4.2.3. Script Execution Method

The recommended method to identify the scripts that must be executed to migrate the DBM tool is the changelog file method (i.e., files to be executed are listed in `install.dbm` and/or `upgrade.dbm` changelog files).

To allow this file to be executed from the `migrate-dbm.sql` script using SQL*Plus, referenced files must all be prefixed with `@@` . SQL*Plus `PROMPT` commands can be used to display useful informative messages.

4.2.4. Home Directory of the DBM Tool

Unlike other tools, which require their work-in-progress material to be stored in a home directory outside of the toolkit, the work-in-progress material for the DBM installer can be hosted entirel, its date, y within the toolkit itself, as long as changes are committed and pushed to Git in a separate branch. This is justified by the fact that modifications could not be easily tested if performed outside of the toolkit. **This is subject to discussion**

SQL scripts are stored in the `sql` subfolder, PL/SQL package code in the `plsql` subfolder, and documentation in the `doc` subfolder. Refer to the SQL Directory, PLSQL Directory, and Doc Directory sections for descriptions of the files they contain.

When the work-in-progress is finalized and released, the files stored in the above subfolders must be copied to the appropriate subfolder under `apps/05_dbm_utility` . Specifically:

- Documentation should be copied to the `doc` subfolder.
- The `ds_objects.sql` script (used to recreate all DB objects) should be copied to the `install` subfolder.
- Package files should be copied to both the `install` and `upgrade` subfolders, depending on the type of release.

4.2.5. Procedure

In addition to the steps described above, all steps outlined in the section Creating a New Version of a Tool must also be followed for the DBM tool.

4.3. Making a new version of the toolkit

Once a new version of a tool has been incorporated in the toolkit, a new bundle (i.e., archive) of the toolkit must be prepared. This also applies when a new version of the DBM tool is released.

The `apps/00_all` folder houses not only the common materials for all tools (such as scripts for exposure, concealment, configuration, and uninstallation of any tool) but also includes a version-specific folder representing the release number of the toolkit bundle.

4.3.1. Assigning a toolkit version number

The toolkit follows the same release number format (`xx.yy[.zz]`) and numbering policy as the individual tools it contains.

The bugfix number `zz` should be used only if the only changes it brings compared to the previous version are related to bug fixing of one or several tools.

4.3.2. Renaming the toolkit version folder

Contrary to individual tools within the toolkit, only one `xx.yy[.zz]` release folder needs to be maintained. The recommended approach is to rename the existing release folder to match the newly

assigned release number.

4.3.3. Updating toolkit release notes

Release notes of the toolkit follow the same format and update practice as those of individual tools except that:

- There is no "New features" section.
- The "Changes" section contains the list of tools and their version that bring new features and/or changes.
- The "Bug fixing" section contains the list of tools and their version that bring bug fixing only.

4.3.4. Building archives

To build the archive (zip file), execute the `mkzip` shell script located in the home directory of the toolkit. This script first executes the `cleanup` shell script that delete temporary files and log files. It then creates an archive named `ec_plsql_toolkit.zip` in the parent directory. This archive must be renamed manually to include the toolkit version and the operating system for which it was created (Windows or Linux). As an example: `ec_plsql_toolkit_24_5_Windows.zip` . Date and time is no more mentioned in the archive name.

An archive must be built for both Windows and Linux platforms. This is in particular necessary for Linux to preserve the execution mode of bash shell scripts.

4.4. Testing archives

It is recommended to test both Windows and Linux archives by unzipping them it onto a file system and initiating the migration process while connected to a database schema that does not already contain the version of the released tools. As a default procedure, uninstall the existing tools, install a previous full release, and then perform the upgrades.

The migration can be launched via the following commands:

- migrate-dbm # upgrade the DBM tool to its latest version (recommended)
- dbm-cli conceal `<tools-list>` # if tools were previously exposed
- dbm-cli migrate `<tools-list>` # migrate tools
- dbm-cli expose `<tools-list>` # re-expose tools if needed

Where `<tools-list>` is a comma separated list of the tools that you want to migrate or `all` for all tools. To be noted that concealing and re-exposing tools is recommended to take care of dropped or newly created database objects.

4.5. Upload archives in Nexus

Upload both archives in the Nexus repository of the DBECoE, under the `ec_plsql_toolkit` folder of the `raw_releases` group.