ARM Utility - User's Guide v24.0.1

Author: Jean-Marc Malmedy (European Commission)

Table of content

1. Generalities

Natively, the creation of ZIP archive files is not supported by Oracle databases. However, there is a need to be able to group and compress several files in a single archive file.

There are several third-party implementations of the ZIP format in PL/SQL. None of them was considered being mature enough to be used in the DBCC developments and flows because of a lack of reliability or maintainability.

Therefore, the Archive Manager internal solution has been developed. It makes it possible to group and compress several files in a single archive file and the solution is based on Oracle native functionalities only. No third-party piece of software is used in this solution.

The solution is based on the *UTL_COMPRESS* native PL/SQL package of Oracle. This package (un)compresses binary or character streams in LZIP format. It is not intended to compress several files in one single archive file. That means that some developments are needed in order to achieve the final goal.

2. Description of the solution

For the creation of a compressed archive file that contains several data (or text) files, the ARM_Utillity works as follows:

1. All data files are concatenated in a temporary binary file.
2. A manifest file is created. This file contains the list of all data files that are concatenated in the temporary binary file and their exact sizes in bytes.
3. The temporary binary files, which is the concatenation of all data files, is compressed using the *UTL_COMPRESS* native PL/SQL package.
4. The temporary binary file is deleted.
5. The compressed archive file and the manifest file must be kept together when the archive needs to be copied or moved and uncompressed in another system.

The manifest file content looks like this:

plain data_file_001.dat:123456
data_file_002.dat:78901
data_file_003.dat:4563478

For the extraction of the data files from a compressed archive file, the ARM_Utility works as follows:

1. The archive file is uncompressed in order to extract the temporary binary file where all data files are concatenated.
2. Using the manifest file containing the data files names and sizes, the data files are extracted from the temporary binary file and recreated with their initial names.
3. The temporary binary file is deleted.
4. A log file can eventually be produced.

3. Technical implementation

3.1. ARM_MAIN_VAR package

This package contains the declaration of all types, variables and exceptions that are used by the packages that implement the logic.

3.1.1. Exceptions

The following exceptions are declared in this package and can be raised by the packages implementing the logic:

Exception	Error code	Description
ge_inv_orcl_dir_name	-20000	Invalid Oracle directory name
ge_inv_src_dir	-20001	Invalid source directory

Exception	Error code	Description
ge_inv_dst_dir	-20002	Invalid destination directory
ge_inv_src_file	-20003	Invalid source file name
ge_inv_dst_file	-20004	Invalid destination file name
ge_inv_file_name	-20005	Invalid file name
ge_inv_dir_name	-20006	Invalid directory name
ge_inv_file_prefix	-20007	Invalid data file prefix
ge_dir_not_exist	-20008	Directory does not exist
ge_file_not_open	-20009	File not open
ge_no_file_to_concat	-20010	No data files to be concatenated
ge_inv_file_size	-20011	Invalid data file size
ge_no_file_to_extract	-20012	No data file to be extracted
ge_inv_manifest_file_name	-20013	Invalid manifest file name
ge_manifest_format	-20014	Invalid manifest file format
ge_dump_file_size_missing	-20015	Data file size missing
ge_src_dir_not_exist	-20016	Source directory does not exist
ge_src_file_not_exist	-20017	Source file does not exist
ge_dst_dir_not_exist	-20018	Destination directory does not exist
ge_orcl_dir_not_exist	-20199 (on AWSimplementation) or -29352 (on Data Center implementation	Oracle directory does not exist

3.1.2. Code generation

Because some application error codes are different on an AWS implementation and a Data Center or a COP implementation, two versions of the *ARM_MAIN_VAR* packages need to be produced.

Those different versions are generated using the GEN_Utility or Code Generator.

A single version of the package is maintained in the *ARM_MAIN_VAR.pks* PL/SQL script. It contains the application error codes for AWS and for the Data Center and the COP. The needed final version of the package, depending on the target environment, is produced using the Code Generator. When the *ENVDC* variable is set, the Code Generator produces the *ARM_MAIN_VAR_DC.pks* PL/SQL script for the Data Center and COP databases. When the *ENVAWS* variable is set, the Code Generator produces the *ARM_MAIN_VAR_AWS.pks* package for the AWS databases.

Because the *ARM_MAIN_VAR* package contains only declarations of types, variables and exceptions, it is only made of a header. This package has no body.

3.2. ARM_UTIL_KRN package

This package implements low level functionalities allowing to retrieve a list of files in an Oracle database directory

Because the implementation is different depending on the environment (an AWS database or a Data Center or COP database), those low-level services are isolated in a dedicated package.

3.2.1. Procedures and functions

3.2.1.1. GET_FILE_NAMES function

This public function returns the list of file names stored in an Oracle directory.

Parameters:

Name	Type	Description
p_directory	VARCHAR2	name of the Oracle directory containing the files
p_file_name_filter	VARCHAR2	an optional name prefix the files must matches in order to be returned

Return value:

Type	Description
nested table of VARCHAR2	the list of names

3.2.2. Code generation

The implementation of the low-level services being different in AWS or Data Center or COP databases, two different versions of the package are needed.

Actually, the package specifications are unique for both environments and are implemented in the *ARM_UTIL_KRN.pks* PL/SQL script.

The package body is maintained in a single *ARM_UTIL_KRN.pkb* PL/SQL script and contains both implementations for the AWS and Data Center or COP environments. The final package, dedicated for one of those environments, is generated using the GEN_Utility or Code Generator.

When the *ENVDC* variable is set, the *ARM_UTIL_KRN_DC.pkb* script is produced. This must be executed in a Data Center or COP database. When the *ENVAWS* variable is set, the *ARM_UTIL_KRN_AWS.pkb* script is produced. This must be executed in an AWS database.

3.3. ARM_MAIN_KRN package

This package is the kernel of the solution and implements its logic.

3.3.1. Procedures and functions

3.3.1.1. RAISE_ERROR procedure

This private procedure adds an error code and the corresponding message to the log file if it is enabled and raises the corresponding application error.

Parameters:

Name	Type	Description
p_code	SIMPLE_INTEGER	application error code
p_message	VARCHAR2	error message

3.3.1.2. LOG_MSG procedure

This procedure adds a message to the log file if it is enabled.

Parameters:

Name	Type	Description
p_message	VARCHAR2	The message to be added to the log file.

3.3.1.3. SET_DIRECTORY procedure

This procedure stores the name of the current Oracle directory. This directory is the one where the data files, archive file, manifest file, …… are stored or generated.

Parameters:

Name	Type	Description
p_directory	VARCHAR2	The directory name

3.3.1.4. INIT_LOG_FILE procedure

This procedure initializes and opens the log file which is optional.

Parameters:

Name	Type	Description
p_log_file_name	VARCHAR2	log file name

3.3.1.5. CLOSE_LOG_FILE procedure

This procedure closes the previously opened log file.

This procedure accepts no parameters.

3.3.1.6. COMPRESS_FILE procedure

This procedure compresses a single file.

Parameters:

Name	Type	Description
p_src_directory	VARCHAR2	Source file directory name
p_src_file	VARCHAR2	Source file name
p_dst_directory	VARCHAR2	Destination file directory name
p_dst_file	VARCHAR2	Destination file name

3.3.1.7. UNCOMPRESS_FILE procedure

This procedure uncompresses a single file.

Parameters=

Name	Type	Description
p_src_directory	VARCHAR2	Compressed file directory name

Name	Type	Description
p_src_file	VARCHAR2	Compressed file name
p_dst_directory	VARCHAR2	Destination file directory name
p_dst_file	VARCHAR2	Destination file name

3.3.1.8. RESET_FILES procedure

This procedure resets the list of files to be concatenated. This list reflects the content of the manifest file.

This procedure accepts no parameters.

3.3.1.9. ADD_FILE procedure

This procedure adds a new item to the list of files to be concatenated.

Parameters:

Name	Type	Description
p_file_name	VARCHAR2	Name of the file to be added
p_file_size	PLS_INTEGER	Size of the file in bytes

3.3.1.10. BUILD_FILE_LIST procedure

This procedure browses a directory in order to identify the items that must be added to the list of files to be concatenated. The scanned directory is the one initialized by the *set_directory* procedure.

Parameters:

Name	Type	Description
p_file_name_prefix	VARCHAR2	The prefix the file names must match in order to be added to the list
p_file_extension	VARCHAR2	An optional extension the file names must match in order to be added to the list

3.3.1.11. CONCAT_FILE function

This private function concatenates a single data file to the already open temporary file.

Parameters:

Name	Type	Description
p_archive	SYS.UTL_FILE.FILE_TYPE	Handle of the archive file the data files must be concatenated in
p_directory	VARCHAR2	Oracle directory containing the data file
p_file_name	VARCHAR2	The name of the data file that must be concatenated in the archive file

Return value:

Type	Description
PLS_INTEGER	Size of the data file in bytes

3.3.1.12. CONCAT_FILE procedure

This private procedure simply calls the *concat_file* function without returning the file size.

Parameters:

Name	Type	Description
p_archive	SYS.UTL_FILE.FILE_TYPE	Handle of the archive file the data files must be concatenated in
p_directory	VARCHAR2	Oracle directory containing the data file
p_file_name	VARCHAR2	The name of the data file that must be concatenated in the archive file

3.3.1.13. CONCAT_FILES procedure

This procedure concatenates the data files in a single file. The data files that are concatenated are the ones that were passed to the *add_file* procedure or that were automatically added by the *build_file_list* procedure.

Parameters:

Name	Type	Description
p_file_name	VARCHAR2	Name of the concatenated file

3.3.1.14. EXTRACT_FILE procedure

This private procedure extracts a single data file from the already open temporary file all data files are concatenated in.

Parameters:

Name	Type	Description
p_archive	SYS.UTL_FILE.FILE_TYPE	Handle of the archive file the data files are concatenated in
p_directory	VARCHAR2	Oracle directory the data file must be extracted in
p_file_name	VARCHAR2	Data file name
p_file_size	PLS_INTEGER	Size of the data file in bytes

3.3.1.15. EXTRACT_FILES procedure

This procedure extracts the data files from the non-compressed concatenated archive file. The data files that are extracted are the ones that were passed to the *add_file* procedure.

Parameters:

Name	Type	Description
p_file_name	VARCHAR2	Name of the concatenated file

3.3.1.16. SAVE_MANIFEST procedure

This procedure saves the manifest file in the directory that was passed to the *set_directory* procedure.

Parameters:

Name	Type	Description
p_manifest_fname	VARCHAR2	Name of the manifest file

3.3.1.17. LOAD_MANIFEST procedure

This procedure loads the manifest file in the directory that was passed to the *set_directory* procedure. A call to the *add_file* procedure is then made for each data file mentioned in the manifest file.

Parameters:

Name	Type	Description

Name	Type	Description
p_manifest_fname	VARCHAR2	Name of the manifest file

3.3.1.18. CREATE_ARCHIVE procedure

This procedure automates the full process of the compressed archive file creation. The procedure calls the lower-level service procedures described above. The produced result is the compressed archive file.

Parameters:

Name	Type	Description
p_directory	VARCHAR2	Directory all files must be read from or created in
p_file_prefix	VARCHAR2	Prefix the data file names must match in order to be archived
p_file_extension	VARCHAR2	Optional extension the data file names must match in order to be archived
p_concat_file	VARCHAR2	Temporary non-compressed concatenated file name
p_archive_file	VARCHAR2	Compressed archive file name
p_manifest_file	VARCHAR2	Manifest file name
p_verbose	BOOLEAN	Optional boolean parameter indicating whether the process must be executed in verbose mode (the DBMS_Output must be activated in order to check the result)
p_log_file_name	VARCHAR2	Optional name of the log file

3.3.1.19. EXTRACT_ARCHIVE_CONTENT procedure

This procedure automates the full process of the data files extraction from the compressed archive file. The procedure calls the lower-level service procedures described above. The produced result is the uncompressed data files.

Parameters:

Name	Type	Description
p_directory	VARCHAR2	Directory all files must be read from or created in
p_archive_file	VARCHAR2	Compressed archive file name

Name	Type	Description
p_manifest_file	VARCHAR2	Manifest file name
p_concat_file	VARCHAR2	Temporary non-compressed concatenated file name
p_verbose	BOOLEAN	Optional boolean parameter indicating whether process must be executed in verbose mode (the DBMS_Output must be activated in order to check the result)
p_log_file_name	VARCHAR2	Optional name of the log file

3.3.2. Code generation

There is a single implementation of the code that is valid for the AWS environments and for the Data Center or COP environments.

The package header is implemented in the *ARM_MAIN_KRN.pks* PL/SQL script and the package body is implemented in the *ARM_MAIN_KRN.pkb* PL/SQL script.

However, the package must be prepared for the execution of the unit test scripts. In order to be able to execute all unit test scripts, the procedures and functions that are tested must be public. That is why the private procedures and functions are declared in the package header for the execution of the unit test scripts. They are removed from the package header afterwards.

The GEN_Utility or Code Generator is used to prepare the package for the unit tests execution. When the code generator is executed with the *UNITTEST* variable set, the private functions and procedures are declared in the package header. They are then made public. When the code generator is executed without this variable, the private procedures and functions are removed from the package header.

3.3.3. Unit tests

The *ARM_MAIN_KRN* package is validated with some unit test scripts. Those tests must be executed with the utPLSQL package.

The unit tests scripts are implemented in the *ARM_MAIN_TST* package. The package header is implemented in the *ARM_MAIN_TST.pks* PL/SQL script and the package body is implemented in the *ARM_MAIN_TST.pkb* PL/SQL script.

4. Using the solution

For the creation of a compressed archive file containing the data files, the *ARM_MAIN_KRN* package can be used manually or in an automated way.

4.1. Compressed archive file creation

4.1.1. Manual use

For the manual use, all the steps are executed by manually invoking the low-level service procedures and functions described above. Those steps and procedure calls are:

1. *set_directory*: This procedure is called once to initialize the directory name all files are read from or created in.
2. *init_log_file*: If a log file needs to be created, this procedure is called once in order to initialize the log file.
3. *reset_files*: This procedure is called once in order to reset the list of data files to be concatenated in a single non-compressed file.
4. Defining the list of files to be concatenated:
   i. *add_file*: This procedure is called for each data file that must be concatenated.
   ii. *build_file_list*: If the data files that must be concatenated can be identified by their name, this procedure can be called once instead of all calls to the *add_file* procedure.
5. *concat_files*: This procedure is called once in order to concatenate all data files defined at the previous step in a single non-compressed file.
6. *compress_file*: This procedure is called once in order the compress the concatenated file produced at the previous step in a compressed archive file.
7. *save_manifest*: This procedure is called once in order to save the manifest file containing the list of data files archived in the compressed file.
8. *close_log_file*: This procedure is called once to close the log file if needed.

4.1.2. Automated use

The *create_archive* procedure is called once in order to create the compressed archive file. It automates the execution of all steps described in the previous chapter.

4.2. Data files extraction

For the extraction of the data files from the compressed archive file, the *ARM_MAIN_KRN* package can also be used manually or in an automated way.

4.2.1. Manual use

For the manual use, all the steps are executed by manually invoking the low-level service procedures and functions described above. Those steps and procedure calls are:

1. *set_directory*: This procedure is called once to initialize the directory name all files are read from or created in.
2. *init_log_file*: If a log file needs to be created, this procedure is called once in order to initialize the log file.
3. *uncompress_file*: This procedure is called once in order to uncompress the compressed archive file to extract the non-compressed concatenated file.
4. *reset_files*: This procedure is called once in order to reset the list of data files to be extracted from the single non-compressed file.
5. Defining the list of files to be extracted:
    i. *add_file*: This procedure is called for each data file that must be extracted. Knowing the name and size of each data file is needed.
    ii. *load_manifest*: This procedure loads the manifest file and calls the *add_file* procedure for each data file mentioned in the manifest file. Manually calling the *add_file* procedure is then not needed anymore.
6. *extract_files*: This procedure is called once in order to extract the data files from the non-compressed concatenated file produced at the previous step.
7. *close_log_file*: This procedure is called once to close the log file if needed.

4.2.2. Automated use

The *extract_archive_content* procedure is called once in order to extract the data files from the compressed archive file. It automates the execution of all steps described in the previous chapter.

5. Installation

5.1. Pre-requisites

The ARM_Utility is a low-level service designed to be integrated or used in any development or higher-level services.

Therefore, it does not depend on any other tool or utility.

5.2. Privileges and roles

5.2.1. System privileges

The database schema user the ARM_Utility needs to be installed in must be granted with the following system privileges:

- *create session*
- *select any dictionary*
- *create table*
- *create view*
- *create trigger*
- *create procedure*
- *create type*
- *unlimited tablespace*

5.2.2. Roles

The database schema user the ARM_Utility needs to be installed in must be granted with the following role, only if the installation environment is a Data Center or a COP (Cloud On Premise) database:

- *javauserpriv*

No special role is required if the installation environment is an AWS database.

5.2.3. Objects privileges

The database schema user the ARM_Utility needs to be installed in must be granted with the execute privilege on the following packages:

- *sys.utl_compress*
- *sys.utl_file*
- *sys.dbms_lob*
- *sys.utl_raw*
- *rdsadmin.rds_file_util*, only if the installation environment is an AWS database.

5.2.4. Database directories

The ARM_Utility reads, creates and deletes files in database directories. The database schema user the ARM_Utility is installed in must then have read and write privileges on all directories archive files must be managed in.

However, the read and write privileges on those directories are not required for the installation of the utility. Those privileges can be granted afterwards. Those privileges are only needed for the use of the utility.

5.3. Installation procedure

The ARM_Utility is now integrated in the **DBCoE PL/SQL Toolkit** and is then deployed and upgraded using the DBM_Utility deployment tool.

The DBM_Utility is able to determinate the type of environment (AWS, Data Center or Cloud on Premise) and automatically deploys the right scripts. No parameters must then be passed to the installer.

Once the **DBCoE PL/SQL Toolkit** is downloaded, the ARM_Utility can be installed using the following commands:

- Deploying the ARM_Utility: `@dbm-cli install arm_utility`
- Migrating to the the last version of the ARM_Utility: `@dbm-cli migrate arm_utility`
- Uninstalling the ARM_Utility: `@dbm-cli uninstall arm_utility`