64 DBMS_DATAPUMP

The DBMS_DATAPUMP package is used to move all, or part of, a database between databases, including both data and metadata.

See Also:

Oracle Database Utilities for more information on the concepts behind the DBMS_DATAPUMP API, how it works, and how it is implemented in the Data Pump Export and Import utilities

This chapter contains the following topics:

- Overview
- Security Model
- Constants
- Data Structures
- Summary of DBMS_DATAPUMP Subprograms

DBMS_DATAPUMP Overview

DBMS_DATAPUMP provides the following support and functionality.

- The source and target databases can have different hardware, operating systems, character sets, time zones, and versions.
- All object types and datatypes existing in Oracle Database 11*g* and higher are supported.
- Data and metadata can be transferred between databases without using any intermediary files.
- A subset of a database can be moved based upon object type and names of objects.
- Schema names, data file names, tablespace names, and data can be transformed at import time.
- Previously aborted export and import jobs can be restarted without duplicating or omitting any data or metadata from the original job.
- The resources applied to an export or import job can be modified.
- Data in an Oracle proprietary format can be unloaded and loaded.

DBMS_DATAPUMP Security Model

Security for the DBMS_DATAPUMP package is implemented through roles.

The DATAPUMP_EXP_FULL_DATABASE and DATAPUMP_IMP_FULL_DATABASE roles enables privileged users to take full advantage of the API. The Oracle Data Pump API will use these roles to determine whether privileged application roles should be assigned to the processes comprising the job.

- `DATAPUMP_EXP_FULL_DATABASE`

  The `DATAPUMP_EXP_FULL_DATABASE` role affects only Export operations. It allows users running these operations to do the following:

  - Perform the operation outside of the scope of their schema
  - Monitor jobs that were initiated by another user
  - Export objects (for example, `TABLESPACE` definitions) that unprivileged users cannot reference

  Although the `SYS` schema does not have the `DATAPUMP_EXP_FULL_DATABASE` role assigned to it, all security checks performed by Data Pump that require the `DATAPUMP_EXP_FULL_DATABASE` role will also grant access to the `SYS` schema.

- `DATAPUMP_IMP_FULL_DATABASE`

  The `DATAPUMP_IMP_FULL_DATABASE` role affects only Import and `SQL_FILE` operations. It allows users running these operations to do the following:

  - Perform the operation outside of the scope of their schema
  - Monitor jobs that were initiated by another user
  - Import objects (for example, `DIRECTORY` definitions) that unprivileged users cannot create

  Although the `SYS` schema does not have the `DATAPUMP_IMP_FULL_DATABASE` role assigned to it, all security checks performed by Oracle Data Pump that require the `DATAPUMP_IMP_FULL_DATABASE` role will also grant access to the `SYS` schema.

DBMS_DATAPUMP Constants

There are several public constants defined for use with the `DBMS_DATAPUMP.GET_STATUS` procedure. All such constants are defined as part of the `DBMS_DATAPUMP` package. Any references to these constants must be prefixed by `DBMS_DATAPUMP.` and followed by the symbols in the following lists:

Mask Bit Definitions

The following mask bit definitions are used for controlling the return of data through the `DBMS_DATAPUMP.GET_STATUS` procedure.

- `KU$_STATUS_WIP CONSTANT BINARY_INTEGER := 1;`
- `KU$_STATUS_JOB_DESC CONSTANT BINARY_INTEGER := 2;`
- `KU$_STATUS_JOB_STATUS CONSTANT BINARY_INTEGER := 4;`
- `KU$_STATUS_JOB_ERROR CONSTANT BINARY_INTEGER := 8;`

Dump File Type Definitions

The following definitions are used for identifying types of dump files returned through the `DBMS_DATAPUMP.GET_STATUS` procedure.

- `KU$_DUMPFILE_TYPE_DISK CONSTANT BINARY_INTEGER := 0;`
- `KU$_DUMPFILE_TYPE_TEMPLATE CONSTANT BINARY_INTEGER := 3;`

DBMS_DATAPUMP Data Structures

The `DBMS_DATAPUMP` package defines `OBJECT` types. The types described in this section are defined in the `SYS` schema for use by the `GET_STATUS` function. *The way in which these types are defined and used may be different than what you are accustomed to. Be sure to read this section carefully.*
The collection of types defined for use with the `GET_STATUS` procedure are version-specific and include version information in the names of the types. Once introduced, these types will always be provided and supported in future versions of Oracle Database and will not change. However, in future releases of Oracle Database, new versions of these types might be created that provide new or different information. The new versions of these types will have different version information embedded in the type names.

For example, in Oracle Database 12c Release 1 (12.1), there is a `sys.ku$_Status1210` type, and in the next Oracle Database release, there can be a `sys.ku$_Status1310` type defined. Both types can be used with the `GET_STATUS` procedure.

Public synonyms have been defined for each of the types used with the `GET_STATUS` procedure. This makes it easier to use the types and means that you do not have to be concerned with changes to the actual type names or schemas where they reside. Oracle recommends that you use these synonyms whenever possible.

For each of the types, there is a version-specific synonym and a generic synonym. For example, the version-specific synonym `ku$_Status1210` is defined for the `sys.ku$_Status1210` type.

The generic synonym always describes the latest version of that type. For example, in Oracle Database 12c, Release 1 (12.1), the generic synonym `ku$_Status` is defined as `ku$_Status1210`. In a future release, there might be a `ku$_Status1310` synonym for `sys.ku$Status1310`. Because the `ku$_Status` generic synonym always points to the latest definition, it would then point to `ku$_Status1310` rather than to `ku$_Status1210`.

The choice of whether to use version-specific synonyms or generic synonyms makes a significant difference in how you work. Using version-specific names protects your code from changes in future releases of Oracle Database because those types will continue to exist and be supported. However, access to new information will require code changes to use new synonym names for each of the types. Using the generic names implies that you always want the latest definition of the types and are prepared to deal with changes in different releases of Oracle Database.

When the version of Oracle Database that you are using changes, any C code that accesses types through generic synonym names will need to be recompiled.

Note:

Languages other than PL/SQL must ensure that their type definitions are properly aligned with the version-specific definitions.

See Also:

GET_STATUS Procedure for additional information about how types are used

Data Structures — Object Types

The `DBMS_DATAPUMP` package defines the following kinds of `OBJECT` types:

- Worker Status Types
- Log Entry and Error Types
- Job Status Types
- Job Description Types
- Status Types

Worker Status Types

The worker status types describe what each worker process in a job is doing. The schema, object name, and object type of an object being processed will be provided. For workers processing user data, the partition name for a partitioned table (if any), the number of bytes processed in the partition, and the number of rows processed in the partition are also returned. Workers processing metadata provide status on the last object that was processed. No status for idle threads is returned.

The `percent_done` refers to the amount completed for the current data item being processed. It is not updated for metadata objects.

The worker status types are defined as follows:

```
CREATE TYPE sys.ku$_WorkerStatus1010 AS OBJECT
        (
                worker_number     NUMBER,       -- Worker process identifier
                process_name      VARCHAR2(30), -- Worker process name
                state             VARCHAR2(30), -- Worker process state
                schema            VARCHAR2(30), -- Schema name
                name              VARCHAR2(4000),-- Object name
                object_type       VARCHAR2(200),-- Object type
                partition         VARCHAR2(30), -- Partition name
                completed_objects NUMBER,       -- Completed number of objects
                total_objects     NUMBER,       -- Total number of objects
                completed_rows    NUMBER,       -- Number of rows completed
                completed_bytes   NUMBER,       -- Number of bytes completed
                percent_done      NUMBER        -- Percent done current object
        )

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatus1010
  FOR sys.ku$_WorkerStatus1010;

CREATE TYPE sys.ku$_WorkerStatus1020 AS OBJECT
        (
                worker_number     NUMBER,       -- Worker process identifier
                process_name      VARCHAR2(30), -- Worker process name
                state             VARCHAR2(30), -- Worker process state
                schema            VARCHAR2(30), -- Schema name
                name              VARCHAR2(4000),-- Object name
                object_type       VARCHAR2(200),-- Object type
                partition         VARCHAR2(30), -- Partition name
                completed_objects NUMBER,       -- Completed number of objects
                total_objects     NUMBER,       -- Total number of objects
                completed_rows    NUMBER,       -- Number of rows completed
                completed_bytes   NUMBER,       -- Number of bytes completed
                percent_done      NUMBER,       -- Percent done current object
                degree          NUMBER        -- Degree of parallelism
        )

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatus1020
  FOR sys.ku$_WorkerStatus1020;

CREATE TYPE sys.ku$_WorkerStatus1120 AS OBJECT
        (
                worker_number     NUMBER,        -- Worker process identifier
                process_name      VARCHAR2(30), -- Worker process name
                state             VARCHAR2(30), -- Worker process state
                schema            VARCHAR2(30), -- Schema name
                name              VARCHAR2(4000),-- Object name
                object_type       VARCHAR2(200),-- Object type
                partition         VARCHAR2(30), -- Partition name
                completed_objects NUMBER,        -- Completed number of objects
                total_objects     NUMBER,        -- Total number of objects
                completed_rows    NUMBER,        -- Number of rows completed
                completed_bytes   NUMBER,        -- Number of bytes completed
                percent_done      NUMBER,        -- Percent done current object
                degree            NUMBER,        -- Degree of parallelism
                instance_id       NUMBER         -- Instance ID where running
        )

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatus1120
  FOR sys.ku$_WorkerStatus1120;

CREATE TYPE sys.ku$_WorkerStatus1210 AS OBJECT
        (
                worker_number     NUMBER,        -- Worker process identifier
                process_name      VARCHAR2(30), -- Worker process name
                state             VARCHAR2(30), -- Worker process state
                schema            VARCHAR2(30), -- Schema name
                name              VARCHAR2(4000),-- Object name
                object_type       VARCHAR2(200),-- Object type
                partition         VARCHAR2(30), -- Partition name
                completed_objects NUMBER,        -- Completed number of objects
                total_objects     NUMBER,        -- Total number of objects
                completed_rows    NUMBER,        -- Number of rows completed
                completed_bytes   NUMBER,        -- Number of bytes completed
                percent_done      NUMBER,        -- Percent done current object
                degree            NUMBER,        -- Degree of parallelism
                instance_id       NUMBER,        -- Instance ID where running
                instance_name     VARCHAR2(60), -- Instance Name where running
                host_name         VARCHAR2(64)  -- Host name where running
        )

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatus1210
  FOR sys.ku$_WorkerStatus1210;

CREATE TYPE sys.ku$_WorkerStatus1220 AS OBJECT
        (
                worker_number     NUMBER,        -- Worker process identifier
                process_name      VARCHAR2(128),-- Worker process name
                state             VARCHAR2(30), -- Worker process state
                schema            VARCHAR2(128),-- Schema name
                name              VARCHAR2(4000),-- Object name
                object_type       VARCHAR2(200),-- Object type
                partition         VARCHAR2(128),-- Partition name
                completed_objects NUMBER,        -- Completed number of objects
                total_objects     NUMBER,        -- Total number of objects
                completed_rows    NUMBER,        -- Number of rows completed
                completed_bytes   NUMBER,       -- Number of bytes completed
                percent_done      NUMBER,       -- Percent done current object
                degree            NUMBER,       -- Degree of parallelism
                instance_id       NUMBER,       -- Instance ID where running
                instance_name     VARCHAR2(60), -- Instance Name where running
                host_name         VARCHAR2(101), -- Host name where running
                access_method     VARCHAR2(16), -- Access Method of object
                obj_start_time    DATE,         -- Object start  time
                obj_status        DATE         -- Object status at current time

        )

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatus1220
  FOR sys.ku$_WorkerStatus1220;

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatus FOR ku$_WorkerStatus1220;

CREATE TYPE sys.ku$_WorkerStatusList1010 AS TABLE OF sys.ku$_WorkerStatus1010
CREATE TYPE sys.ku$_WorkerStatusList1020 AS TABLE OF sys.ku$_WorkerStatus1020
CREATE TYPE sys.ku$_WorkerStatusList1120 AS TABLE OF sys.ku$_WorkerStatus1120
CREATE TYPE sys.ku$_WorkerStatusList1210 AS TABLE OF sys.ku$_WorkerStatus1210
CREATE TYPE sys.ku$_WorkerStatusList1220 AS TABLE OF sys.ku$_WorkerStatus1220

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatusList1010
  FOR sys.ku$_WorkerStatusList1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatusList1020
  FOR sys.ku$_WorkerStatusList1020;
CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatusList1120
  FOR sys.ku$_WorkerStatusList1120;
CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatusList1210
  FOR sys.ku$_WorkerStatusList1210;
CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatusList1220
  FOR sys.ku$_WorkerStatusList1220;

CREATE OR REPLACE PUBLIC SYNONYM ku$_WorkerStatusList
  FOR ku$_WorkerStatusList1220;
```

Log Entry and Error Types

These types provide informational and error text to attached clients and the log stream. The `ku$LogLine.errorNumber` type is set to `NULL` for informational messages but is specified for error messages. Each log entry may contain several lines of text messages.

The log entry and error types are defined as follows:

```
CREATE TYPE sys.ku$_LogLine1010 AS OBJECT (
               logLineNumber   NUMBER,
               errorNumber     NUMBER,
               LogText         VARCHAR2(2000))

CREATE OR REPLACE PUBLIC SYNONYM ku$_LogLine1010 FOR sys.ku$_LogLine1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_LogLine1020 FOR sys.ku$_LogLine1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_LogLine FOR ku$_LogLine1010;
CREATE TYPE sys.ku$_LogEntry1010 AS TABLE OF sys.ku$_LogLine1010

CREATE OR REPLACE PUBLIC SYNONYM ku$_LogEntry1010 FOR sys.ku$_LogEntry1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_LogEntry1020 FOR sys.ku$_LogEntry1010;
CREATE OR REPLACE PUBLIC SYNONYM ku$_LogEntry FOR ku$_LogEntry1010;
```

Job Status Types

The job status type returns status about a job. Usually, the status concerns a running job, but it could also be about a stopped job when a client attaches. It is typically requested at attach time, when the client explicitly requests status from interactive mode and every *N* seconds when the client has requested status periodically.

The job status types are defined as follows (`percent_done` applies to data only):

```
CREATE TYPE sys.ku$_JobStatus1010 IS OBJECT
        (
                job_name        VARCHAR2(30),           -- Name of the job
                operation       VARCHAR2(30),           -- Current operation
                job_mode        VARCHAR2(30),           -- Current mode
                bytes_processed NUMBER,                 -- Bytes so far
                total_bytes     NUMBER,                 -- Total bytes for job
                percent_done    NUMBER,                 -- Percent done
                degree          NUMBER,                 -- Of job parallelism
                error_count     NUMBER,                 -- #errors so far
                state           VARCHAR2(30),           -- Current job state
                phase           NUMBER,                 -- Job phase
                restart_count   NUMBER,                 -- #Job restarts
                worker_status_list ku$_WorkerStatusList1010, -- For (non-idle)
                                                        -- job worker processes
                files           ku$_DumpFileSet1010     -- Dump file info
        )

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobStatus1010 FOR sys.ku$_JobStatus1010;

CREATE TYPE sys.ku$_JobStatus1020 IS OBJECT
        (
                job_name        VARCHAR2(30),           -- Name of the job
                operation       VARCHAR2(30),           -- Current operation
                job_mode        VARCHAR2(30),           -- Current mode
                bytes_processed NUMBER,                 -- Bytes so far
                total_bytes     NUMBER,                 -- Total bytes for job
                percent_done    NUMBER,                 -- Percent done
                degree          NUMBER,                 -- Of job parallelism
                error_count     NUMBER,                 -- #errors so far
                state           VARCHAR2(30),           -- Current job state
                phase           NUMBER,                 -- Job phase
                restart_count   NUMBER,                 -- #Job restarts
                worker_status_list ku$_WorkerStatusList1020, -- For (non-idle)
                                                        -- job worker processes
                files           ku$_DumpFileSet1010     -- Dump file info
        )

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobStatus1020 FOR sys.ku$_JobStatus1020;

CREATE TYPE sys.ku$_JobStatus1120 IS OBJECT
        (
                job_name        VARCHAR2(30),           -- Name of the job
                operation       VARCHAR2(30),           -- Current operation
                job_mode        VARCHAR2(30),           -- Current mode
                bytes_processed NUMBER,                 -- Bytes so far
                total_bytes     NUMBER,                 -- Total bytes for job
                percent_done    NUMBER,                 -- Percent done
                degree          NUMBER,                 -- Of job parallelism
                error_count     NUMBER,                 -- #errors so far
                state           VARCHAR2(30),           -- Current job state
                phase           NUMBER,                 -- Job phase
                restart_count   NUMBER,                 -- #Job restarts
                worker_status_list ku$_WorkerStatusList1120, -- For (non-idle)
                                                        -- job worker processes
                    files           ku$_DumpFileSet1010     -- Dump file info
        )

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobStatus1120 FOR sys.ku$_JobStatus1120;

CREATE TYPE sys.ku$_JobStatus1210 IS OBJECT
        (
                job_name        VARCHAR2(30),           -- Name of the job
                operation       VARCHAR2(30),           -- Current operation
                job_mode        VARCHAR2(30),           -- Current mode
                bytes_processed NUMBER,                 -- Bytes so far
                total_bytes     NUMBER,                 -- Total bytes for job
                percent_done    NUMBER,                 -- Percent done
                degree          NUMBER,                 -- Of job parallelism
                error_count     NUMBER,                 -- #errors so far
                state           VARCHAR2(30),           -- Current job state
                phase           NUMBER,                 -- Job phase
                restart_count   NUMBER,                 -- #Job restarts
                worker_status_list ku$_WorkerStatusList1210, -- For (non-idle)
                                                        -- job worker processes
                files           ku$_DumpFileSet1010     -- Dump file info
        )

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobStatus1210 FOR sys.ku$_JobStatus1210;

CREATE TYPE sys.ku$_JobStatus1220 IS OBJECT
        (
                job_name        VARCHAR2(128),          -- Name of the job
                operation       VARCHAR2(30),           -- Current operation
                job_mode        VARCHAR2(30),           -- Current mode
                bytes_processed NUMBER,                 -- Bytes so far
                total_bytes     NUMBER,                 -- Total bytes for job
                percent_done    NUMBER,                 -- Percent done
                degree          NUMBER,                 -- Of job parallelism
                error_count     NUMBER,                 -- #errors so far
                state           VARCHAR2(30),           -- Current job state
                phase           NUMBER,                 -- Job phase
                restart_count   NUMBER,                 -- #Job restarts
                heartbeat       NUMBER,                 -- Job heartbeat
                worker_status_list ku$_WorkerStatusList1220, -- For (non-idle)
                                                        -- job worker processes
                files           ku$_DumpFileSet1010     -- Dump file info
        )


CREATE OR REPLACE PUBLIC SYNONYM ku$_JobStatus1220 FOR sys.ku$_JobStatus1220;

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobStatus FOR ku$_JobStatus1220;
```

Job Description Types

The job description type holds all the environmental information about the job such as parameter settings and dump file set members. There are a couple of subordinate types required as well.

The job description types are defined as follows:

```
CREATE TYPE sys.ku$_JobDesc1010 IS OBJECT
        (
                job_name        VARCHAR2(30),           -- The job name
                guid            RAW(16),                -- The job GUID
                operation       VARCHAR2(30),           -- Current operation
                job_mode        VARCHAR2(30),           -- Current mode
                remote_link     VARCHAR2(4000),         -- DB link, if any
                owner           VARCHAR2(30),           -- Job owner
                instance        VARCHAR2(16),           -- The instance name
                db_version      VARCHAR2(30),           -- Version of objects
                creator_privs   VARCHAR2(30),           -- Privs of job
                start_time      DATE,                   -- This job start time
                max_degree      NUMBER,                 -- Max. parallelism
                log_file        VARCHAR2(4000),         -- Log file name
                sql_file        VARCHAR2(4000),         -- SQL file name
                params          ku$_ParamValues1010     -- Parameter list
        )

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobDesc1010 FOR sys.ku$_JobDesc1010;

CREATE TYPE sys.ku$_JobDesc1020 IS OBJECT
        (
                job_name        VARCHAR2(30),           -- The job name
                guid            RAW(16),                -- The job GUID
                operation       VARCHAR2(30),           -- Current operation
                job_mode        VARCHAR2(30),           -- Current mode
                remote_link     VARCHAR2(4000),         -- DB link, if any
                owner           VARCHAR2(30),           -- Job owner
                platform        VARCHAR2(101),          -- Current job platform
                exp_platform    VARCHAR2(101),          -- Export platform
                global_name     VARCHAR2(4000),         -- Current global name
                exp_global_name VARCHAR2(4000),         -- Export global name
                instance        VARCHAR2(16),           -- The instance name
                db_version      VARCHAR2(30),           -- Version of objects
                exp_db_version  VARCHAR2(30),           -- Export version
                scn             NUMBER,                 -- Job SCN
                creator_privs   VARCHAR2(30),           -- Privs of job
                start_time      DATE,                   -- This job start time
                exp_start_time  DATE,                   -- Export start time
                term_reason     NUMBER,                 -- Job termination code
                max_degree      NUMBER,                 -- Max. parallelism
                log_file        VARCHAR2(4000),         -- Log file name
                sql_file        VARCHAR2(4000),         -- SQL file name
                params          ku$_ParamValues1010     -- Parameter list
        )

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobDesc1020 FOR sys.ku$_JobDesc1020;

CREATE TYPE sys.ku$_JobDesc1210 IS OBJECT
        (
                job_name        VARCHAR2(30),           -- The job name
                guid            RAW(16),                -- The job GUID
                operation       VARCHAR2(30),           -- Current operation
                job_mode        VARCHAR2(30),           -- Current mode
                remote_link     VARCHAR2(4000),         -- DB link, if any
                owner           VARCHAR2(30),           -- Job owner
                platform        VARCHAR2(101),          -- Current job platform
                exp_platform    VARCHAR2(101),          -- Export platform
                global_name     VARCHAR2(4000),         -- Current global name
                exp_global_name VARCHAR2(4000),         -- Export global name
                instance        VARCHAR2(16),           -- The instance name
                db_version      VARCHAR2(30),           -- Cur. server software version
                exp_db_version  VARCHAR2(30),           -- Export svr. software version
                job_version     VARCHAR2(30),           -- Negotiated data version
                scn             NUMBER,                 -- Job SCN
                creator_privs   VARCHAR2(30),           -- Privs of job
                start_time      DATE,                   -- This job start time
                exp_start_time  DATE,                   -- Export start time
                term_reason     NUMBER,                 -- Job termination code
                max_degree      NUMBER,                 -- Max. parallelism
                timezone        VARCHAR2(64),           -- Cur. server timezone
                exp_timezone    VARCHAR2(64),           -- Exp. server timezone
                tstz_version    NUMBER,                 -- Cur. server timezone version
                exp_tstz_version NUMBER,                -- Exp. server timezone
                endianness      VARCHAR2(16),           -- Cur. platform's endianness
                exp_endianness  VARCHAR2(16),           -- Exp. platform's endianness
-- endianness is 'BIG' or 'LITTLE'
                charset         VARCHAR2(28),           -- Cur. server charset
                exp_charset     VARCHAR2(28),           -- Exp. server charset
                ncharset        VARCHAR2(28),           -- Cur. server national charset
                exp_ncharset    VARCHAR2(28),           -- Exp. server national charset
                log_file        VARCHAR2(4000),         -- Log file name
                sql_file        VARCHAR2(4000),         -- SQL file name
                params          ku$_ParamValues1010     -- Parameter list
        )

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobDesc1210 FOR sys.ku$_JobDesc1210;

CREATE TYPE sys.ku$_JobDesc1220 IS OBJECT
        (
                job_name        VARCHAR2(128),          -- The job name
                guid            RAW(16),                -- The job GUID
                operation       VARCHAR2(30),           -- Current operation
                job_mode        VARCHAR2(30),           -- Current mode
                remote_link     VARCHAR2(4000),         -- DB link, if any
                owner           VARCHAR2(128),          -- Job owner
                platform        VARCHAR2(101),          -- Current job platform
                exp_platform    VARCHAR2(101),          -- Export platform
                global_name     VARCHAR2(4000),         -- Current global name
                exp_global_name VARCHAR2(4000),         -- Export global name
                instance        VARCHAR2(60),           -- The instance name
                db_version      VARCHAR2(60),           -- Cur. server software version
                exp_db_version  VARCHAR2(60),           -- Export svr. software version
                job_version     VARCHAR2(60),           -- Negotiated data version
                scn             NUMBER,                 -- Job SCN
                creator_privs   VARCHAR2(30),           -- Privs of job
                start_time      DATE,                   -- This job start time
                exp_start_time  DATE,                   -- Export start time
                term_reason     NUMBER,                 -- Job termination code
                max_degree      NUMBER,                 -- Max. parallelism
                timezone        VARCHAR2(64),           -- Cur. server timezone
                exp_timezone    VARCHAR2(64),           -- Exp. server timezone
                tstz_version    NUMBER,                 -- Cur. server timezone version
                exp_tstz_version NUMBER,                -- Exp. server timezone
                endianness      VARCHAR2(16),           -- Cur. platform's endianness
                exp_endianness  VARCHAR2(16),           -- Exp. platform's endianness
-- endianness is 'BIG' or 'LITTLE'
                charset         VARCHAR2(28),           -- Cur. server charset
                exp_charset     VARCHAR2(28),           -- Exp. server charset
                ncharset        VARCHAR2(28),           -- Cur. server national charset
                exp_ncharset    VARCHAR2(28),           -- Exp. server national charset
                log_file        VARCHAR2(4000),         -- Log file name
                sql_file        VARCHAR2(4000),         -- SQL file name
                params          ku$_ParamValues1010     -- Parameter list
        )

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobDesc1220 FOR sys.ku$_JobDesc1220;

CREATE OR REPLACE PUBLIC SYNONYM ku$_JobDesc FOR ku$_JobDesc1220;
```

Status Types

The status type is an aggregate of some the previous types defined and is the return value for the `GET_STATUS` call. The mask attribute indicates which types of information are being returned to the caller. It is created by a client's shadow process from information it retrieves off the status queue or directly from the master table.

For errors, the `ku$_LogEntry` that is returned has already had its log lines ordered for proper output. That is, the original `ku$_LogEntry` objects have been ordered from outermost context to innermost.

The status types are defined as follows:

```
CREATE TYPE sys.ku$_Status1010 IS OBJECT
        (
                mask            NUMBER,            -- Status types present
                wip             ku$_LogEntry1010, -- Work in progress
                job_description ku$_JobDesc1010,  -- Complete job description
                job_status      ku$_JobStatus1010,-- Detailed job status
                error           ku$_LogEntry1010  -- Multi-level context errors
        )

CREATE OR REPLACE PUBLIC SYNONYM ku$_Status1010 FOR sys.ku$_Status1010;

CREATE TYPE sys.ku$_Status1020 IS OBJECT
        (
                mask            NUMBER,            -- Status types present
                wip             ku$_LogEntry1010, -- Work in progress
                job_description ku$_JobDesc1020,  -- Complete job description
                job_status      ku$_JobStatus1020,-- Detailed job status
                error           ku$_LogEntry1010  -- Multi-level context errors
        )

CREATE OR REPLACE PUBLIC SYNONYM ku$_Status1020 FOR sys.ku$_Status1020;

CREATE TYPE sys.ku$_Status1120 IS OBJECT
        (
                mask            NUMBER,            -- Status types present
                wip             ku$_LogEntry1010, -- Work in progress
                job_description ku$_JobDesc1020,  -- Complete job description
                job_status      ku$_JobStatus1120,-- Detailed job status
                error           ku$_LogEntry1010  -- Multi-level context errors
        )

CREATE OR REPLACE PUBLIC SYNONYM ku$_Status1120 FOR sys.ku$_Status1120;

CREATE TYPE sys.ku$_Status1210 IS OBJECT
        (
                mask            NUMBER,            -- Status types present
                wip             ku$_LogEntry1010, -- Work in progress
                job_description ku$_JobDesc1210,  -- Complete job description
                job_status      ku$_JobStatus1210,-- Detailed job status
                error           ku$_LogEntry1010  -- Multi-level context errors
        )

CREATE OR REPLACE PUBLIC SYNONYM ku$_Status1210 FOR sys.ku$_Status1210;
CREATE TYPE sys.ku$_Status1220 IS OBJECT
        (
                mask            NUMBER,            -- Status types present
                wip             ku$_LogEntry1010, -- Work in progress
                job_description ku$_JobDesc1220,  -- Complete job description
                job_status      ku$_JobStatus1220,-- Detailed job status
                error           ku$_LogEntry1010  -- Multi-level context errors
        )

CREATE OR REPLACE PUBLIC SYNONYM ku$_Status1220 FOR sys.ku$_Status1220;

CREATE OR REPLACE PUBLIC SYNONYM ku$_Status FOR ku$_Status1220;
```

Summary of DBMS_DATAPUMP Subprograms

This table lists the DBMS_DATAPUMP subprograms in alphabetical order and briefly describes them.

Table 64-1 DBMS_DATAPUMP Package Subprograms

Subprogram	Description
ADD_FILE Procedure	Adds dump files to the dump file set for an Export, Import, or SQL_FILE operation. In addition to dump files, other types of files can also be added by using the FILETYPE parameter provided with this procedure.
ATTACH Function	Used to gain access to a Data Pump job that is in the Defining, Executing, Idling, or Stopped state
DATA_FILTER Procedures	Specifies restrictions on the rows that are to be retrieved
DATA_REMAP Procedure	Specifies transformations to be applied to column data as it is exported from, or imported into, a database.
DETACH Procedure	Specifies that the user has no further interest in using the handle
GET_DUMPFILE_INFO Procedure	Retrieves information about a specified dump file
GET_STATUS Procedure	Monitors the status of a job or waits for the completion of a job or for more details on API errors
LOG_ENTRY Procedure	Inserts a message into the log file
METADATA_FILTER Procedure	Provides filters that allow you to restrict the items that are included in a job
METADATA_REMAP Procedure	Specifies a remapping to be applied to objects as they are processed in the specified job
METADATA_TRANSFORM Procedure	Specifies transformations to be applied to objects as they are processed in the specified job
OPEN Function	Declares a new job using the Data Pump API, the handle returned being used as a parameter for calls to all other procedures (but not to the ATTACH function)
SET_PARALLEL Procedure	Adjusts the degree of parallelism within a job
SET_PARAMETER Procedures	Specifies job-processing options
START_JOB Procedure	Begins or resumes execution of a job
STOP_JOB Procedure	Terminates a job, but optionally, preserves the state of the job
WAIT_FOR_JOB Procedure	Runs a job until it either completes normally or stops for some other reason

ADD_FILE Procedure

This procedure adds files to the dump file set for an Export, Import, or `SQL_FILE` operation, or specifies the log file or the output file for a `SQL_FILE` operation.

Syntax

```
DBMS_DATAPUMP.ADD_FILE (
   handle      IN NUMBER,
   filename    IN VARCHAR2,
   directory   IN VARCHAR2,
   filesize    IN VARCHAR2 DEFAULT NULL,
   filetype    IN NUMBER DEFAULT DBMS_DATAPUMP.KU$_FILE_TYPE_DUMP_FILE,
   reusefile   IN NUMBER DEFAULT NULL);
```

Parameters

Table 64-2 ADD_FILE Procedure Parameters

Parameter	Description
`handle`	The handle of a job. The current session must have previously attached to the handle through a call to either the `OPEN` or `ATTACH` function.
`filename`	The name of the file that is being added. The `filename` parameter must be a simple filename without any directory path information. For dump files, the `filename` can include a substitution variable. For a description of available substitution variables, see the following table. The file can be written to or read from either the local file system or the Oracle Object Store. If you are interfacing with the local file system, the `filename` parameter must contain a filename without directory path information. If you are interfacing with the Oracle Object Store, the `filename` parameter must contain a valid `URI` to the location of a bucket within your compartment in the Oracle Object Store. You must also specify a valid `CREDENTAL` in the directory parameter to access the bucket and the `DBMS_DATAPUMP.KU$_FILE_TYPE_URIDUMP_FILE` file type in the filetype parameter. For examples, see "Using the Oracle Data Pump API" in *Oracle Database Utilities*.
`directory`	If you are interfacing with the local file system, then the directory parameter specifies the name of a directory object within the database that is used to locate the filename. If you are interfacing with the Oracle Object Store, then the `directory` parameter must specify a valid `CREDENTIAL` to access the bucket where filename resides. Ensure that you specify a valid `directory` parameter.

Table 64-2 (Cont.) ADD_FILE Procedure Parameters

Parameter	Description
`filesize`	The size of the dump file that is being added. It can be specified as follows: • The number of bytes • The number of kilobytes (if followed by `K`) • The number of megabytes (if followed by `M`) • The number of gigabytes (if followed by `G`) • The number of terabytes (if followed by `T`) An Export operation will write no more than the specified number of bytes to the file. When the file is full, it will be closed. If there is insufficient space on the device to write the specified number of bytes, then the `Export` operation will fail, but it can be restarted. If not specified, then `filesize` defaults to an unlimited size. For `Import` and `SQL_FILE` operations, `filesize` is ignored. The minimum value for `filesize` is ten times the default Data Pump block size, which is `4` kilobytes. A `filesize` can only be specified for dump files.
`filetype`	The type of the file that you want to add. The supported values are as follows: • `DBMS_DATAPUMP.KU$_FILE_TYPE_DUMP_FILE`—Dump file for a job • `DBMS_DATAPUMP.KU$_FILE_TYPE_LOG_FILE`—Log file for a job • `DBMS_DATAPUMP.KU$_FILE_TYPE_SQL_FILE`—Output for `SQL_FILE` job • `DBMS_DATAPUMP.KU$_FILE_TYPE_URIDUMP_FILE`—Specifies a filename in the Oracle Object Store. For more information, see the `filename` and `directory` parameters described above.
`reusefile`	If the value is `0`, then a preexisting file will cause an error. If the value is `1`, then a preexisting file will be overwritten. If the value is `NULL`, then the default action for the file type will be applied (that is, dump files will not be overwritten). This parameter should only be non-`NULL` for dump files. The `reusefile` parameter is restricted to export jobs.

Substitution Variables

Table 64-3 Substitution Variables Available for the Filename Parameter on DBMS_DATAPUMP.ADD_FILE

Substitution Variable	Description
`%U`	The `%U` is expanded in the resulting file names into a two-character, fixed-width, incrementing integer starting at 01, and ending at 99. For example, the dump filename of `export%U` would cause `export01`, `export02`, `export03`, and so on, to be created depending on how many files are needed to perform the export up to `export99`. For filenames containing the `%` character, the `%` must be represented as `%%` to avoid ambiguity. Note: If you have more than 100 files, then Oracle recommends that you use the `%L` substitution variable.

Table 64-3 (Cont.) Substitution Variables Available for the Filename Parameter on DBMS_DATAPUMP.ADD_FILE

Substitution Variable	Description

`%l`, `%L`	Specifies a system-generated unique file name.
	The file names can contain a substitution variable (`%L`), which implies that multiple files may be generated. The substitution variable is expanded in the resulting file names into a 2-digit, fixed-width, incrementing integer starting at 01 and ending at 99 which is the same as (`%U`). In addition, the substitution variable is expanded in the resulting file names into a 3-digit to 10-digit, variable-width, incrementing integers starting at 100 and ending at 2147483646. The width field is determined by the number of digits in the integer.
	For example if the current integer was 1, `exp%Laa%L.dmp` would resolve to
	`exp01aa01.dmp`
	`exp02aa02.dmp`
	and so forth up until 99. Then, the next file name would have 3 digits substituted:
	`exp100aa100.dmp`
	`exp101aa101.dmp`
	and so forth up until 999 where the next file would have 4 digits substituted. The substitution will continue up to the largest number substitution allowed, which is 2147483646.

`%d`, `%D`	Specifies the current day of the month from the Gregorian calendar in format `DD`. Note: This substitution variable cannot be used in an import file name.
`%m`, `%M`	Specifies the month in the Gregorian calendar in format `MM`. Note: This substitution variable cannot be used in an import file name.
`%t`, `%T`	Specifies the year, month, and day in the Gregorian calendar in this format: `YYYYMMDD`. Note: This substitution variable cannot be used in an import file name.
`%y`, `%Y`	Specifies the year in this format: YYYY. Note: This substitution variable cannot be used in an import file name.

Exceptions

- `INVALID_HANDLE`. The specified handle is not attached to an Oracle Data Pump job.
- `INVALID_ARGVAL`. An invalid value was supplied for an input parameter.
- `INVALID_STATE`. The job is completing, or the job is past the defining state for an import or `SQL_FILE` job, or is past the defining state for `LOG` and `SQL` files.
- `INVALID_OPERATION`. A dump file was specified for a Network Import or `ESTIMATE_ONLY` export operation.
- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.
- `NO_SUCH_JOB`. The specified job does not exist.

Usage Notes

- This procedure adds files to an Oracle Data Pump job. You can add the following types of files to a job:
  - Log files—To record the messages associated with an operation. The Log file overwrites the previously existing files.
  - SQL files—To record the output of a `SQL_FILE` operation. The SQL file overwrites the previously existing files.
  - Dump files—To contain the data that is being moved. The Dump files do not overwrite the existing files. However, an error is generated.
- Import and `SQL_FILE` operations require that that you specify all dump files during the definition phase of the job. For Export operations, dump files can be added at any time. For example, if the user ascertains that the file space is running low during an Export, additional dump files can be added through this API. If the specified dump file already exists for an Export operation and `reusefile` is not set to 1, an error will be returned.
- For Export operations, the parallelism setting should be less than or equal to the number of dump files in the dump file set. If there are not enough dump files, then the job will not be able to maximize parallelism to the degree specified by the `SET_PARALLEL` procedure.
- For Import operations, the parallelism setting should also be less than or equal to the number of dump files in the dump file set. If there are not enough dump files, the performance will not be optimal, as multiple threads of execution try to access the same dump file.
- If the substitution variable (`%U`) is included in a filename, then multiple dump files can be specified through a single call to `ADD_FILE`. For Export operations, the new dump files will be created as they are needed. Enough dump files will be created to allow all of the processes specified by the current `SET_PARALLEL` value to be active. If one of the dump files is full, then it is closed, and a new dump file (with a new generated name) is created to take its place. If multiple `ADD_FILE`s with substitution variables have been specified for dump files in a job, then they will be used to generate dump files in a round-robin fashion. For example, if `expa%U`, `expb%U` and `expc%U` were all specified for a job having a parallelism of 6, then the initial dump files created would appear as follows: `expa01`, `expb01`, `expc01`, `expa02`, `expb02`, and `expc02`.
- If presented with dump file specifications, `expa%U`, `expb%U` and `expc%U`, then an Import or `SQL_FILE` operation will begin by attempting to open the dump files, `expa01`, `expb01`, and `expc01`. If the dump file containing the master table is not found in this set, then the operation will expand its search for dump files by incrementing the substitution variable and looking up the new filenames (for example, `expa02`, `expb02`, and `expc02`). The Oracle Data Pump API will keep expanding the search until it locates the dump file containing the master table. If the Oracle Data Pump API determines that the dump file does not exist, or that it is not part of the current dump set at any iteration, then the Oracle Data Pump API stops incrementing the substitution variable for the dump file specification that was in error. After the master table is found, the master table is used to ascertain when all of dump files in the dump file set have been located.

Examples

> **Note:**
>
> The examples in this section assume that the credentials, network ACLs, database account, and object-store information are already set up.

Example 1

The following example performs a table mode export to the Oracle Object Store.

```
CONNECT user;
Enter password: password

SET SERVEROUTPUT ON
SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100

DECLARE
  hdl           NUMBER;           -- Datapump handle
  ind           NUMBER;           -- Loop index
  le            ku$_LogEntry;     -- For WIP and error messages
  js            ku$_JobStatus;    -- The job status from get_status
  jd            ku$_JobDesc;      -- The job description from get_status
  sts           ku$_Status;       -- The status object returned by get_status
  jobState      VARCHAR2(30);     -- To keep track of job state
  dumpFile      VARCHAR2(1024)    := 'https://example.oraclecloud.com/test/
den02ten_foo3b_split_%u.dat';
  dumpType      NUMBER            := dbms_datapump.ku$_file_type_uridump_file;
  credName      VARCHAR2(1024)    := 'BMCTEST';
  logFile       VARCHAR2(1024)    := 'tkopc_export3b_cdb2.log';
  logDir        VARCHAR2(9)       := 'WORK';
  logType       NUMBER            := dbms_datapump.ku$_file_type_log_file;

BEGIN

  --
  -- Open a schema-based export job and perform defining-phase initialization.
  --
  hdl := dbms_datapump.open('EXPORT', 'TABLE');
  dbms_datapump.set_parameter(hdl, 'COMPRESSION', 'ALL');
  dbms_datapump.set_parameter(hdl, 'CHECKSUM', 1);
  dbms_datapump.add_file(hdl, logfile, logdir, null, logType);
  dbms_datapump.add_file(hdl, dumpFile, credName, '3MB', dumpType, 1);
  dbms_datapump.data_filter(hdl, 'INCLUDE_ROWS', 1);
  dbms_datapump.metadata_filter(hdl, 'TABLE_FILTER', 'FOO', '');
  --
  -- Start the job.
```

```
--
dbms_datapump.start_job(hdl);

--
-- Now grab output from the job and write to standard out.
--
jobState := 'UNDEFINED';
WHILE (jobState != 'COMPLETED') AND (jobState != 'STOPPED')
LOOP
  dbms_datapump.get_status(hdl,
         dbms_datapump.ku$_status_job_error +
         dbms_datapump.ku$_status_job_status +
         dbms_datapump.ku$_status_wip, -1, jobState,sts);
  js := sts.job_status;

  --
  -- If we received any WIP or Error messages for the job, display them.
  --
  IF (BITAND(sts.mask,dbms_datapump.ku$_status_wip) != 0) THEN
    le := sts.wip;
  ELSE
    IF (bitand(sts.mask,dbms_datapump.ku$_status_job_error) != 0) THEN
      le := sts.error;
    ELSE
      le := NULL;
    END IF;
  END IF;

  IF le IS NOT NULL THEN
    ind := le.FIRST;
    WHILE ind IS NOT NULL LOOP
      dbms_output.put_line(le(ind).LogText);
      ind := le.NEXT(ind);
    END LOOP;
  END IF;
END LOOP;

--
-- Detach from job.
--
dbms_datapump.detach(hdl);

--
-- Any exceptions that propagated to this point will be captured.
-- The details are retrieved from get_status and displayed.
--
EXCEPTION
  WHEN OTHERS THEN
    BEGIN
      dbms_datapump.get_status(hdl, dbms_datapump.ku$_status_job_error, 0,
                               jobState, sts);
      IF (BITAND(sts.mask,dbms_datapump.ku$_status_job_error) != 0) THEN
        le := sts.error;
        IF le IS NOT NULL THEN
          ind := le.FIRST;
          WHILE ind IS NOT NULL LOOP
```

```
              dbms_output.put_line(le(ind).LogText);
              ind := le.NEXT(ind);
            END LOOP;
          END IF;
        END IF;

      BEGIN
        dbms_datapump.stop_job (hdl, 1, 0, 0);
      EXCEPTION
        WHEN OTHERS THEN NULL;
      END;

      EXCEPTION
      WHEN OTHERS THEN
        dbms_output.put_line('Unexpected exception while in exception ' ||
                             'handler. sqlcode = ' || TO_CHAR(SQLCODE));
      END;
END;
/
EXIT;
```

Example 2

The follwoing example performs a table mode import from the Oracle Object Store.

```
CONNECT user;
Enter password: password

SET SERVEROUTPUT ON
SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100

DECLARE
  hdl          NUMBER;         -- Datapump handle
  ind          NUMBER;         -- Loop index
  le           ku$_LogEntry;   -- For WIP and error messages
  js           ku$_JobStatus;  -- The job status from get_status
  jd           ku$_JobDesc;    -- The job description from get_status
  sts          ku$_Status;     -- The status object returned by get_status
  jobState     VARCHAR2(30);   -- To keep track of job state
  dumpFile     VARCHAR2(1024)  := 'https://example.oraclecloud.com/test/
den02ten_foo3b_split_%u.dat';
  dumpType     NUMBER          := dbms_datapump.ku$_file_type_uridump_file;
  credName     VARCHAR2(1024)  := 'BMCTEST';
  logFile      VARCHAR2(1024)  := 'tkopc_import3b_cdb2.log';
  logDir       VARCHAR2(9)     := 'WORK';
  logType      NUMBER          := dbms_datapump.ku$_file_type_log_file;

BEGIN

  --
```

```
-- Open a schema-based export job and perform defining-phase initialization.
--
hdl := dbms_datapump.open('IMPORT', 'TABLE', NULL, 'OSI');
dbms_datapump.add_file(hdl, logfile, logdir, null, logType);
dbms_datapump.add_file(hdl, dumpFile, credName, null, dumpType);
dbms_datapump.metadata_filter(hdl, 'TABLE_FILTER', 'FOO', '');
dbms_datapump.set_parameter(hdl, 'TABLE_EXISTS_ACTION', 'REPLACE');
dbms_datapump.set_parameter(hdl, 'VERIFY_CHECKSUM', 1);

--
-- Start the job.
--
dbms_datapump.start_job(hdl);

--
-- Now grab output from the job and write to standard out.
--
jobState := 'UNDEFINED';
WHILE (jobState != 'COMPLETED') AND (jobState != 'STOPPED')
LOOP
  dbms_datapump.get_status(hdl,
         dbms_datapump.ku$_status_job_error +
         dbms_datapump.ku$_status_job_status +
         dbms_datapump.ku$_status_wip, -1, jobState,sts);
  js := sts.job_status;

  --
  -- If we received any WIP or Error messages for the job, display them.
  --
  IF (BITAND(sts.mask,dbms_datapump.ku$_status_wip) != 0) THEN
    le := sts.wip;
  ELSE
    IF (bitand(sts.mask,dbms_datapump.ku$_status_job_error) != 0) THEN
      le := sts.error;
    ELSE
      le := NULL;
    END IF;
  END IF;

  IF le IS NOT NULL THEN
    ind := le.FIRST;
    WHILE ind IS NOT NULL LOOP
      dbms_output.put_line(le(ind).LogText);
      ind := le.NEXT(ind);
    END LOOP;
  END IF;
END LOOP;

--
-- Detach from job.
--
dbms_datapump.detach(hdl);

--
-- Any exceptions that propagated to this point will be captured.
-- The details are retrieved from get_status and displayed.
```

```
  --
  EXCEPTION
    WHEN OTHERS THEN
      BEGIN
        dbms_datapump.get_status(hdl, dbms_datapump.ku$_status_job_error, 0,
                                 jobState, sts);
        IF (BITAND(sts.mask,dbms_datapump.ku$_status_job_error) != 0) THEN
          le := sts.error;
          IF le IS NOT NULL THEN
            ind := le.FIRST;
            WHILE ind IS NOT NULL LOOP
              dbms_output.put_line(le(ind).LogText);
              ind := le.NEXT(ind);
            END LOOP;
          END IF;
        END IF;

      BEGIN
        dbms_datapump.stop_job (hdl, 1, 0, 0);
      EXCEPTION
        WHEN OTHERS THEN NULL;
      END;

      EXCEPTION
      WHEN OTHERS THEN
        dbms_output.put_line('Unexpected exception while in exception ' ||
                             'handler. sqlcode = ' || TO_CHAR(SQLCODE));
      END;
END;
/
EXIT;
```

ATTACH Function

This function provides access to a previously created job.

Syntax

```
DBMS_DATAPUMP.ATTACH(
   job_name     IN VARCHAR2 DEFAULT NULL,
   job_owner     IN VARCHAR2 DEFAULT NULL)
 RETURN NUMBER;
```

Parameters

Table 64-4 ATTACH Function Parameters

Parameter	Description
`job_name`	The name of the job. The default is the job name owned by the user who is specified in the `job_owner` parameter (assuming that user has only one job in the Defining, Executing, or Idling states).

Table 64-4 (Cont.) ATTACH Function Parameters

Parameter	Description
job_owner	The user who originally started the job. If NULL, then the value defaults to the owner of the current session. To specify a job owner other than yourself, you must have either the DATAPUMP_EXP_FULL_DATABASE role (for export operations) or the DATAPUMP_IMP_FULL_DATABASE role (for import and SQL_FILE operations). Being a privileged user allows you to monitor another user's job, but you cannot restart another user's job.

Return Values

An opaque handle for the job. This handle is used as input to the following procedures: ADD_FILE, DATA_FILTER, DETACH, GET_STATUS, LOG_ENTRY, METADATA_FILTER, METADATA_REMAP, METADATA_TRANSFORM, SET_PARALLEL, SET_PARAMETER,START_JOB, STOP_JOB, and WAIT_FOR_JOB.

Exceptions

- INVALID_ARGVAL. An invalid value was supplied for an input parameter.
- OBJECT_NOT_FOUND. The specified job no longer exists or the user specified a job owned by another schema, but the user did not have the DATAPUMP_EXP_FULL_DATABASE or DATAPUMP_IMP_FULL_DATABASE role.
- SUCCESS_WITH_INFO. The function succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

- If the job was in the Stopped state, then the job is placed into the Idling state. After the ATTACH succeeds, you can monitor the progress of the job or control the job. The stream of KU$_STATUS_WIP and KU$_STATUS_JOB_ERROR messages returned through the GET_STATUS procedure will be returned to the newly attached job starting at the approximate time of the client's attachment. There will be no repeating of status and error messages that were processed before the client attached to a job.
- If you want to perform a second attach to a job, then you must do so from a different session.
- If the ATTACH fails, then use a null handle in a subsequent call to GET_STATUS for more information about the failure.

DATA_FILTER Procedures

This procedure specifies restrictions on the rows that are to be retrieved.

Syntax

```
DBMS_DATAPUMP.DATA_FILTER (
   handle      IN NUMBER,
   name        IN VARCHAR2,
   value       IN NUMBER,
   table_name  IN VARCHAR2 DEFAULT NULL,
   schema_name IN VARCHAR2 DEFAULT NULL);
```

```
DBMS_DATAPUMP.DATA_FILTER(
   handle      IN NUMBER,
   name        IN VARCHAR2,
   value       IN VARCHAR2,
   table_name  IN VARCHAR2 DEFAULT NULL,
   schema_name IN VARCHAR2 DEFAULT NULL);

DBMS_DATAPUMP.DATA_FILTER(
   handle      IN NUMBER,
   name        IN VARCHAR2,
   value       IN CLOB,
   table_name  IN VARCHAR2 DEFAULT NULL,
   schema_name IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 64-5 DATA_FILTER Procedure Parameters

Parameter	Description
`handle`	The handle that is returned from the `OPEN` function
`name`	The name of the filter
`value`	The value of the filter
`table_name`	The name of the table on which the data filter is applied. If no table name is supplied, the filter applies to all tables in the job.
`schema_name`	The name of the schema that owns the table on which the filter is applied. If no schema name is specified, the filter applies to all schemas in the job. If you supply a schema name you must also supply a table name.

Exceptions

- `INVALID_ARGVAL`. There can be several reasons for this message:
  - A bad filter name is specified
  - The mode is `TRANSPORTABLE`, which does not support data filters
  - The specified table does not exist
  - The filter has already been set for the specified values of `schema_name` and `table_name`
- `INVALID_STATE`. The user called `DATA_FILTER` when the job was not in the Defining state.
- `INCONSISTENT_ARGS`. The `value` parameter is missing or its datatype does not match the filter name. Or a schema name was supplied, but not a table name.
- `PRIVILEGE_ERROR`. A schema name was supplied, but the user did not have the `DATAPUMP_EXP_FULL_DATABASE` or `DATAPUMP_IMP_FULL_DATABASE` role.
- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.
- `NO_SUCH_JOB`. The specified job does not exist.

Usage Notes

- Each data filter can only appear once in each table (for example, you cannot supply multiple `SUBQUERY` filters to a table) or once in each job. If different filters using the same name are applied to both a particular table and to the whole job, the filter parameter supplied for the specific table will take precedence.

With the exception of the `INCLUDE_ROWS` filter, data filters are not supported on tables having nested tables or domain indexes defined upon them. Data filters are not supported in jobs performed in Transportable Tablespace mode.

The available data filters are described in Table 64-6.

Table 64-6 Data Filters

Name	Datatype	Operations that Support Filter	Description
`INCLUDE_ROWS`	`NUMBER`	EXPORT, IMPORT	If nonzero, this filter specifies that user data for the specified table should be included in the job. The default is 1.
`PARTITION_EXPR` `PARTITION_LIST`	`TEXT`	EXPORT, IMPORT	**Note:** In this description, the information about partitions also applies to subpartions. For Export jobs, these filters specify which partitions are unloaded from the database. For Import jobs, they specify which table partitions are loaded into the database. Partition names are included in the job if their names satisfy the specified expression (for `PARTITION_EXPR`) or are included in the list (for `PARTITION_LIST`). Whereas the expression version of the filter offers more flexibility, the list version provides for full validation of the partition names. Double quotation marks around partition names are required only if the partition names contain special characters. `PARTITION_EXPR` is not supported on jobs across a network link. Default=All partitions are processed
`SAMPLE`	`NUMBER`	EXPORT, IMPORT	For Export jobs, specifies a percentage for sampling the data blocks to be moved. This filter allows subsets of large tables to be extracted for testing purposes.
`SUBQUERY`	`TEXT`	EXPORT, IMPORT	Specifies a subquery that is added to the end of the `SELECT` statement for the table. If you specify a `WHERE` clause in the subquery, you can restrict the rows that are selected. Specifying an `ORDER BY` clause orders the rows dumped in the export which improves performance when migrating from heap-organized tables to index-organized tables.

DATA_REMAP Procedure

This procedure specifies transformations to be applied to column data as it is exported from, or imported into, a database.

Syntax

```
DBMS_DATAPUMP.DATA_REMAP(
   handle          IN NUMBER,
   name            IN VARCHAR2,
   table_name      IN VARCHAR2,
   column          IN VARCHAR2,
   remap_function  IN VARCHAR2,
   schema          IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 64-7 DATA_REMAP Procedure Parameters

Parameter	Description
`handle`	The handle of the current job. The current session must have previously attached to the handle through a call to an `OPEN` function.
`name`	The name of the remap
`table_name`	The table containing the column to be remapped
`column`	The name of the column to be remapped
`remap_function`	The meaning of remap_function is dependent upon the value of name. See Table 64-8 for a list of possible names.
`schema`	The schema containing the column to be remapped. If NULL, the remapping applies to all schemas moved in the job that contain the specified table.

Exceptions

- `INVALID_ARGVAL`. The mode is transportable (which does not support data modifications) or it has specified that no data to be included in the job. An invalid remap name was supplied.
- `INVALID_OPERATION`. Data remaps are only supported for Export and Import operations.
- `INVALID_STATE`. The `DATA_REMAP` procedure was called after the job started (that is, it was not in the defining state).
- `NO_SUCH_JOB`. The job handle is no longer valid.

Usage Notes

- The `DATA_REMAP` procedure is only supported for Export and Import operations. It allows you to manipulate user data being exported or imported. The `name` of the remap determines the remap operation to be performed.
- For export operations, you might wish to define a data remap to obscure sensitive data such as credit card numbers from a dump file, but leave the remainder of the data so that it can be read. To accomplish this, the remapping should convert each unique source number into a distinct generated number. So that the mapping is consistent across the dump file set, the same function should be called for every column that contains the credit card number.

- For import operations, you might wish to define a data remap to reset the primary key when data is being merged into an existing table that contains colliding primary keys. A single remapping function should be provided for all columns defining or referencing the primary key to ensure that remapping is consistent.

Note:

If the called function uses package state variables, then to ensure that remapping is performed consistently across all tables, the job should be run with a SET_PARALLEL value of 1 and no restart operations should be performed.

The Data Remap functions are listed in Table 64-8.

Table 64-8 Names of Data Remap Functions

Name	Meaning of `remap_function`	Meaning
`COLUMN_FUNCTION`	String having the format: `[schema.]package.function`	The `name` parameter references a PL/SQL package function which is called to modify the data for the specified column. The function accepts a single parameter, which has the same datatype as the remapped column, and returns a value having the same datatype as the remapped column. Note that the default for the schema is the schema of the user performing the export.

DETACH Procedure

This procedure specifies that the user has no further interest in using the handle.

Syntax

```
DBMS_DATAPUMP.DETACH(
   handle   IN NUMBER);
```

Parameters

Table 64-9 DETACH Procedure Parameters

Parameter	Description
`handle`	The handle of the job. The current session must have previously attached to the handle through a call to either an `OPEN` or `ATTACH` function.

Exceptions

- `INVALID_HANDLE`. The specified handle is not attached to a Data Pump job.
- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.

- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

- Through this call, you specify that you have no further interest in using the handle. Resources associated with a completed job cannot be reclaimed until all users are detached from the job. An implicit detach from a handle is performed when the user's session is exited or aborted. An implicit detach from a handle is also performed upon the expiration of the timeout associated with a STOP_JOB that was applied to the job referenced by the handle. All previously allocated DBMS_DATAPUMP handles are released when an instance is restarted.

GET_DUMPFILE_INFO Procedure

This procedure retrieves information about a specified dump file.

Syntax

```
DBMS_DATAPUMP.GET_DUMPFILE_INFO(
   filename     IN VARCHAR2,
   directory    IN VARCHAR2,
   info_table   OUT ku$_dumpfile_info,
   filetype     OUT NUMBER);
```

Parameters

Table 64-10 GET_DUMPFILE_INFO Procedure Parameters

Parameter	Description
filename	A simple filename with no directory path information
directory	A directory object that specifies where the file can be found
info_table	A PL/SQL table for storing information about the dump file
filetype	The type of file (Data Pump dump file, original Export dump file, external tables dump file, or unknown)

Exceptions

The GET_DUMPFILE_INFO procedure is a utility routine that operates outside the context of any Data Pump job. Exceptions are handled differently for this procedure than for procedures associated in some way with a Data Pump job. A full exception stack should be available directly, without the need to call the GET_STATUS procedure to retrieve the detailed information. The exception for this procedure is as follows:

- NO_DUMPFILE_INFO. Unable to retrieve dump file information as specified.

Usage Notes

You can use the GET_DUMPFILE_INFO procedure to request information about a specific file. If the file is not recognized as any type of dump file, then a filetype of 0 (zero) is returned and the dump file info_table remains empty.

A filetype value of 1 indicates a Data Pump dump file. A filetype value of 2 indicates an original Export dump file. A filetype value of 3 indicates an external tables dump file. In all cases, the dump file info_table will be populated with information retrieved from the dump file header. Rows of this table consist of item code and value pairs, where the item code indicates the type of information and the value column is a VARCHAR2 containing the actual data (converted to a string in some cases). The table is defined as follows:

```
CREATE TYPE sys.ku$_dumpfile_item IS OBJECT (
                item_code       NUMBER,           -- Identifies header item
                value           VARCHAR2(2048)    -- Text string value)/

GRANT EXECUTE ON sys.ku$_dumpfile_item TO PUBLIC;
CREATE OR REPLACE PUBLIC SYNONYM ku$_dumpfile_item FOR sys.ku$_dumpfile_item;

CREATE TYPE sys.ku$_dumpfile_info AS TABLE OF sys.ku$_dumpfile_item/

GRANT EXECUTE ON sys.ku$_dumpfile_info TO PUBLIC;
CREATE OR REPLACE PUBLIC SYNONYM ku$_dumpfile_info FOR sys.ku$_dumpfile_info;
```

The item codes, which can easily be extended to provide more information as needed, are currently defined as shown in Table 64-11 (prepended with the package name, DBMS_DATAPUMP.). Assume the following with regard to these item codes:

- Unless otherwise stated, all item codes may be returned only for Oracle Data Pump and external tables dump files (filetypes 1 and 3).
- Unless otherwise stated, all item codes have been available since Oracle Database 10*g* Release 2 (10.2).

Table 64-11 Item Codes For the DBMS_DATAPUMP.GET_DUMPFILE_INFO Procedure

Item Code	Description
KU$_DFHDR_FILE_VERSION	The internal file version of the dump file.
KU$_DFHDR_MASTER_PRESENT	If the Data Pump master table is present in the dump file, then the value for this item code is 1; otherwise the value is 0. Returned only for filetype 1.
KU$_DFHDR_GUID	A unique identifier assigned to the Data Pump export job or the external tables unload job that produced the dump file. For a multifile dump set, each file in the set has the same value for this item code.
KU$_DFHDR_FILE_NUMBER	A numeric identifier assigned to the dump file. Each dump file in a multifile dump set has its own identifier, unique only within the dump set.
KU$_DFHDR_CHARSET_ID	A numeric code that represents the character set in use at the source system when the dump file was created. Returned for all filetypes.
KU$_DFHDR_CREATION_DATE	The date and time that the dump file was created.
KU$_DFHDR_FLAGS	Internal flag values.
KU$_DFHDR_JOB_NAME	The name assigned to the export job that created the dump file. Returned only for filetype 1.
KU$_DFHDR_PLATFORM	The operating system name of the source system on which the dump file was created.
KU$_DFHDR_INSTANCE	The instance name of the source system on which the dump file was created.
KU$_DFHDR_LANGUAGE	The language name that corresponds to the character set of the source system where the export dump file was created.
KU$_DFHDR_BLOCKSIZE	The blocksize, in bytes, of the dump file.
KU$_DFHDR_DIRPATH	If direct path mode was used when the dump file was created, then the value for this item code is 1, otherwise the value is 0. Returned only for filetype 2.

Table 64-11 (Cont.) Item Codes For the DBMS_DATAPUMP.GET_DUMPFILE_INFO Procedure

Item Code	Description
`KU$_DFHDR_METADATA_COMPRESSED`	If the system metadata is stored in the dump file in compressed format, then the value for this item code is 1, otherwise the value is 0. Returned only for filetype 1.
`KU$_DFHDR_DB_VERSION`	The database job version used to create the dump file. Returned for all filetypes.
`KU$_DFHDR_MASTER_PIECE_COUNT`	The Data Pump master table may be split into multiple pieces and written to multiple dump files in the set, one piece per file. The value returned for this item code indicates the number of dump files that contain pieces of the master table. The value for this item code is only meaningful if the Data Pump master table is present in the dump file, as indicated by the item code `KU$_DFHDR_MASTER_PRESENT`. Returned only for filetype 1. Only available since Oracle Database 11*g* Release 1 (11.1).
`KU$_DFHDR_MASTER_PIECE_NUMBER`	The Data Pump master table may be split into multiple pieces and written to multiple dump files in the set, one piece per file. The value returned for this item code indicates which master table piece is contained in the dump file. The value for this item code is only meaningful if the Data Pump master table is present in the dump file, as indicated by the item code `KU$_DFHDR_MASTER_PRESENT`. Returned only for filetype 1. Only available since Oracle Database 11*g* Release 1 (11.1).
`KU$_DFHDR_DATA_COMPRESSED`	If the table data is stored in the dump file in compressed format, then the value for this item code is 1, otherwise the value is 0. Only available since Oracle Database 11*g* Release 1 (11.1).
`KU$_DFHDR_METADATA_ENCRYPTED`	If the system metadata is stored in the dump file in encrypted format, then the value for this item code is 1, otherwise the value is 0. Returned only for filetype 1. Only available since Oracle Database 11*g* Release 1 (11.1).
`KU$_DFHDR_DATA_ENCRYPTED`	If the table data is stored in the dump file in encrypted format, then the value for this item code is 1, otherwise the value is 0. Only available since Oracle Database 11*g* Release 1 (11.1).
`KU$_DFHDR_COLUMNS_ENCRYPTED`	If encrypted column data is stored in the dump file in encrypted format, then the value for this item code is 1, otherwise the value is 0. Returned only for filetype 1. Only available since Oracle Database 11*g* Release 1 (11.1).

Table 64-11 (Cont.) Item Codes For the DBMS_DATAPUMP.GET_DUMPFILE_INFO Procedure

Item Code	Description
`KU$_DFHDR_ENCRYPTION_MODE`	The encryption mode indicates whether a user-provided password or the Oracle Encryption Wallet was used to encrypt data written to the dump file. The possible values returned for this item code are: • `KU$_DFHDR_ENCMODE_NONE` No data was written to the dump file in encrypted format. • `KU$_DFHDR_ENCMODE_PASSWORD` Data was written to the dump file in encrypted format using a provided password. • `KU$_DFHDR_ENCMODE_DUAL` Data was written to the dump file in encrypted format using both a provided password as well as an Oracle Encryption Wallet. • `KU$_DFHDR_ENCMODE_TRANS` Data was written to the dump file in encrypted format transparently using an Oracle Encryption Wallet. Only available since Oracle Database 11*g* Release 1 (11.1).
`KU$_DFHDR_COMPRESSION_ALG`	The compression algorithm used when writing system metadata and/or table data to the dump file in compressed format. The possible values returned for this item code are: • `KU$_DFHDR_CMPALG_NONE` No data was written to the dump file in compressed format. • `KU$_DFHDR_CMPALG_BASIC` Data was written to the dump file in compressed format using an internal algorithm. This is the default algorithm used since Oracle Database 10*g* Release 2 (10.2). • `KU$_DFHDR_CMPALG_LOW` Data was written to the dump file in compressed format using the `LOW` algorithm. • `KU$_DFHDR_CMPALG_MEDIUM` Data was written to the dump file in compressed format using the `MEDIUM` algorithm. • `KU$_DFHDR_CMPALG_HIGH` Data was written to the dump file in compressed format using the `HIGH` algorithm. Only available since Oracle Database 12*c* Release 1 (12.1).

GET_STATUS Procedure

This procedure monitors the status of a job or waits for the completion of a job.

Syntax

```
DBMS_DATAPUMP.GET_STATUS(
   handle     IN NUMBER,
   mask       IN BINARY_INTEGER,
   timeout    IN NUMBER DEFAULT NULL,
   job_state OUT VARCHAR2,
   status    OUT ku$_Status);
```

Parameters

Table 64-12 GET_STATUS Procedure Parameters

Parameter	Description
`handle`	The handle of a job. The current session must have previously attached to the handle through a call to either the `OPEN` or `ATTACH` function. A null handle can be used to retrieve error information after `OPEN` and `ATTACH` failures.
`mask`	A bit mask that indicates which of four types of information to return: • `KU$_STATUS_WIP` • `KU$_STATUS_JOB_DESC` • `KU$_STATUS_JOB_STATUS` • `KU$_STATUS_JOB_ERROR` Each status has a numerical value. You can request multiple types of information by adding together different combinations of values.
`timeout`	Maximum number of seconds to wait before returning to the user. A value of 0 requests an immediate return. A value of -1 requests an infinite wait. If `KU$_STATUS_WIP` or `KU$_STATUS_JOB_ERROR` information is requested and becomes available during the timeout period, then the procedure returns before the timeout period is over.
`job_state`	Current state of the job. If only the job state is needed, it is much more efficient to use this parameter than to retrieve the full `ku$_Status` structure.
`status`	A ku$_Status is returned. The ku$_Status mask indicates what kind of information is included. This could be none if only `KU$_STATUS_WIP` or `KU$_STATUS_JOB_ERROR` information is requested and the timeout period expires.

Exceptions

- `INVALID_HANDLE`. The specified handle is not attached to a Data Pump job.
- `INVALID_VALUE`. The mask or timeout contains an illegal value.
- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.
- `NO_SUCH_JOB`. The specified job does not exist.

Usage Notes

The `GET_STATUS` procedure is used to monitor the progress of an ongoing job and to receive error notification. You can request various type of information using the mask parameter. The `KU$_STATUS_JOB_DESC` and `KU$_STATUS_JOB_STATUS` values are classified as synchronous information because the information resides in the master table. The `KU$_STATUS_WIP` and `KU$_STATUS_JOB_ERROR` values are classified as asynchronous because the messages that embody these types of information can be generated at any time by various layers in the Data Pump architecture.

- If synchronous information *`only`* is requested, the interface will ignore the timeout parameter and simply return the requested information.
- If asynchronous information is requested, the interface will wait a *`maximum`* of timeout seconds before returning to the client. If a message of the requested asynchronous information type is received, the call will complete prior to timeout seconds. If synchronous information was also requested, it will be returned whenever the procedure returns.

- If the `job_state` returned by `GET_STATUS` does not indicate a terminating job, it is possible that the job could still terminate before the next call to `GET_STATUS`. This would result in an `INVALID_HANDLE` exception. Alternatively, the job could terminate during the call to `GET_STATUS`, which would result in a `NO_SUCH_JOB` exception. Callers should be prepared to handle these cases.

Error Handling

There are two types of error scenarios that need to be handled using the `GET_STATUS` procedure:

- Errors resulting from other procedure calls: For example, the `SET_PARAMETER` procedure may produce an `INCONSISTENT_ARGS` exception. The client should immediately call `GET_STATUS` with `mask=8` (errors) and `timeout=0`. The returned ku$_Status.error will contain a ku$_LogEntry that describes the inconsistency in more detail.
- Errors resulting from events asynchronous to the client(s): An example might be `Table already exists` when trying to create a table. The `ku$_Status.error` will contain a `ku$_LogEntry` with all error lines (from all processing layers that added context about the error) properly ordered.

After a job has begun, a client's main processing loop will typically consist of a call to `GET_STATUS` with an infinite timeout (-1) "listening" for `KU$_STATUS_WIP` and `KU$_STATUS_JOB_ERROR` messages. If status was requested, then `JOB_STATUS` information will also be in the request.

When the ku$_Status is interpreted, the following guidelines should be used:

- `ku$_Status.ku$_JobStatus.percent_done` refers only to the amount of data that has been processed in a job. Metadata is not considered in the calculation. It is determined using the following formulas:
  - EXPORT or network IMPORT--`(bytes_processed/estimated_bytes) * 100`
  - IMPORT--`(bytes_processed/total_expected_bytes) * 100`
  - SQL_FILE or estimate-only EXPORT--`0.00` if not done or `100.00` if done

  The effects of the `QUERY` and `PARTITION_EXPR` data filters are not considered in computing `percent_done`.

  It is expected that the status returned will be transformed by the caller into more user-friendly status. For example, when percent done is not zero, an estimate of completion time could be produced using the following formula:

  ```
  ((SYSDATE - start time) / ku$_Status.ku$_JobStatus.percent_done) * 100
  ```
- The caller should not use `ku$_Status.ku$_JobStatus.percent_done` for determining whether the job has completed. Instead, the caller should only rely on the state of the job as found in `job_state`.

LOG_ENTRY Procedure

This procedure inserts a message into the log file.

Syntax

```
DBMS_DATAPUMP.LOG_ENTRY(
   handle         IN NUMBER,
   message        IN VARCHAR2
   log_file_only  IN NUMBER DEFAULT 0);
```

Parameters

Table 64-13 LOG_ENTRY Procedure Parameters

Parameter	Description
`handle`	The handle of a job. The current session must have previously attached to the handle through a call to either the `OPEN` or `ATTACH` function.
`message`	A text line to be added to the log file
`log_file_only`	Specified text should be written only to the log file. It should not be returned in `GET_STATUS` work-in-progress (`KU$_STATUS_WIP`) messages.

Exceptions

- `INVALID_HANDLE`. The specified handle is not attached to a Data Pump job.
- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.
- `NO_SUCH_JOB`. The specified job does not exist.

Usage Notes

The message is added to the log file. If `log_file_only` is zero (the default), the message is also broadcast as a `KU$_STATUS_WIP` message through the `GET_STATUS` procedure to all users attached to the job.

The `LOG_ENTRY` procedure allows applications to tailor the log stream to match the abstractions provided by the application. For example, the command-line interface supports `INCLUDE` and `EXCLUDE` parameters defined by the user. Identifying these values as calls to the underlying `METADATA_FILTER` procedure would be confusing to users. Instead, the command-line interface can enter text into the log describing the settings for the `INCLUDE` and `EXCLUDE` parameters.

Lines entered in the log stream from `LOG_ENTRY` are prefixed by the string, "`;;; `"

METADATA_FILTER Procedure

This procedure provides filters that allow you to restrict the items that are included in a job.

Syntax

```
DBMS_DATAPUMP.METADATA_FILTER(
   handle       IN NUMBER,
   name         IN VARCHAR2,
   value        IN VARCHAR2,
   object_path  IN VARCHAR2 DEFAULT NULL);

DBMS_DATAPUMP.METADATA_FILTER(
   handle       IN NUMBER,
   name         IN VARCHAR2,
   value        IN CLOB,
   object_path  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 64-14 METADATA_FILTER Procedure Parameters

Parameter	Description
`handle`	The handle returned from the `OPEN` function
`name`	The name of the filter. See Table 64-15 for descriptions of the available filters.
`value`	The value of the filter
`object_path`	The object path to which the filter applies. If the default is used, the filter applies to all applicable objects. Lists of the object paths supported for each mode are contained in the catalog views for `DATABASE_EXPORT_OBJECTS`, `SCHEMA_EXPORT_OBJECTS`, and `TABLE_EXPORT_OBJECTS`. (Note that the `TABLE_EXPORT_OBJECTS` view is applicable to both Table and Tablespace mode because their object paths are the same.) For an import operation, object paths reference the mode used to create the dump file rather than the mode being used for the import.

Table 64-15 describes the name, the object type, and the meaning of the filters available with the `METADATA_FILTER` procedure. The datatype for all the filters is a text expression. All operations support all filters.

Table 64-15 Filters Provided by METADATA_FILTER Procedure

Name	Object Type	Meaning
`NAME_EXPR` `NAME_LIST`	Named objects	Defines which object names are included in the job. You use the object type parameter to limit the filter to a particular object type. For Table mode, identifies which tables are to be processed.
`SCHEMA_EXPR` `SCHEMA_LIST`	Schema objects	Restricts the job to objects whose owning schema name is satisfied by the expression. For Table mode, only a single `SCHEMA_EXPR` filter is supported. If specified, it must only specify a single schema (for example, `'IN (''SCOTT'')'`). For Schema mode, identifies which users are to be processed.
`TABLESPACE_EXPR` `TABLESPACE_LIST`	`TABLE`, `CLUSTER`, `INDEX`, `ROLLBACK_SEGMENT`	Restricts the job to objects stored in a tablespace whose name is satisfied by the expression. For Tablespace mode, identifies which tablespaces are to be processed. If a partition of an object is stored in the tablespace, the entire object is added to the job. For Transportable mode, identifies which tablespaces are to be processed. If a table has a single partition in the tablespace set, all partitions must be in the tablespace set. An index is not included within the tablespace set unless all of its partitions are in the tablespace set. A domain index is not included in the tablespace set unless all of its secondary objects are included in the tablespace set.

Table 64-15 (Cont.) Filters Provided by METADATA_FILTER Procedure

Name	Object Type	Meaning

`INCLUDE_PATH_EXPR`
`INCLUDE_PATH_LIST`
`EXCLUDE_PATH_EXPR`
`EXCLUDE_PATH_LIST`
	All	Defines which object paths are included in, or excluded from, the job. You use these filters to select only certain object types from the database or dump file set. Objects of paths satisfying the condition are included (`INCLUDE_PATH_*`) or excluded (`EXCLUDE_PATH_*`) from the operation. The `object_path` parameter is not supported for these filters.

`EXCLUDE_TABLES`	`TABLE_EXPORT`	Specifies that no tables are to be exported. The `EXCLUDE_TABLES` parameter is needed only when `VIEWS_AS_TABLES` is specified, and when there are no tables in a table mode export.
`VIEWS_AS_TABLES`	`TABLE_EXPORT`	A comma-separated list of views that you want to be exported as tables: [*schema_name*.]*view_name*[:*table_name*] The filter can be called multiple times with multiple values. All values are added to a list. All views on the list are exported as tables.

Exceptions

- `INVALID_HANDLE`. The specified handle is not attached to a Data Pump job.
- `INVALID_ARGVAL`. This exception can indicate any of the following conditions:
  - An `object_path` was specified for an `INCLUDE_PATH_EXPR` or `EXCLUDE_PATH_EXPR` filter.
  - The specified `object_path` is not supported for the current mode.
  - The `SCHEMA_EXPR` filter specified multiple schemas for a Table mode job.
- `INVALID_STATE`. The user called the `METADATA_FILTER` procedure after the job left the defining state.
- `INCONSISTENT_ARGS`. The filter value is of the wrong datatype, or the filter is missing.
- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.
- `NO_SUCH_JOB`. The specified job does not exist.

Usage Notes

- Metadata filters identify a set of objects that you want to be included or excluded from an Oracle Data Pump operation. Except for `EXCLUDE_PATH_EXPR` and `INCLUDE_PATH_EXPR`, dependent objects of an identified object will be processed along with the identified object. For example, if an index is identified for inclusion by a filter, then grants on that index will also be included by the filter. Likewise, if a table is excluded by a filter, then indexes, constraints, grants and triggers on the table will also be excluded by the filter.
- Two versions of each filter are supported: SQL expression and List. The SQL expression version of the filters offer maximum flexibility for identifying objects (for example the use of `LIKE` to support use of wild cards). The names of the expression filters are as follows:

  - `NAME_EXPR`
  - `SCHEMA_EXPR`
  - `TABLESPACE_EXPR`
  - `INCLUDE_PATH_EXPR`
  - `EXCLUDE_PATH_EXPR`

  The list version of the filters allow maximum validation of the filter. An error will be reported if one of the elements in the filter is not found within the source database (for Export and network-based jobs) or is not found within the dump file (for file-based Import and SQLFILE jobs). The names of the list filters are as follows:

  - `NAME_LIST`
  - `SCHEMA_LIST`
  - `TABLESPACE_LIST`
  - `INCLUDE_PATH_LIST`
  - `EXCLUDE_PATH_LIST`

- Filters allow a user to restrict the items that are included in a job. For example, a user could request a full export, but without Package Specifications or Package Bodies.
- If multiple filters are specified for a object type, they are implicitly 'ANDed' together (that is, objects participating in the job must pass all of the filters applied to their object types).
- The same filter name can be specified multiple times within a job. For example, specifying `NAME_EXPR as '!=''EMP'''` and `NAME_EXPR as '!=''DEPT'''` on a Table mode export would produce a file set containing all of the tables except for `EMP` and `DEPT`.
- `EXCLUDE_TABLES` is only used in concert with `VIEWS_AS_TABLES`, so that you can exclude tables from an export while enabling exports of views as tables. For example, if the only thing you want to do is to export the contents of a view as a table, then you can use `VIEWS_AS_TABLES` to export the views, but not the tables. If you want to export some actual tables, and you also want export other views as tables, then you would select `VIEWS_AS_TABLES`, but not select `EXCLUDE_TABLES`.

DBMS_DATAPUMP Examples

In the following example, we use the metadata filters `EXCLUDE_TABLES` with `VIEWS_AS_TABLES` to export a view as a table, while excluding the table from which the view is created.

Suppose you have a view called `SCOTT.EMPV` that is defined for a specific set of columns in the table `SCOTT.EMP` (for example, only the columns `empno,ename` and `hiredate`). You want to export that view as a table to a target database using the Oracle Data Pump API, so that you only expose that subset of columns from `SCOTT.EMP` in your export. In this case, you can run the export as follows:

```
BEGIN
  h := dbms_datapump.open('EXPORT' ,'TABLE');
  dbms_datapump.metadata_filter(h, 'EXCLUDE_TABLES', 'Y');
  dbms_datapump.metadata_filter(h, 'VIEWS_AS_TABLES','SCOTT.EMPV');
:
:
:
 dbms_datapump.start_job(h, 0, 0);
```

```
:
:
```

If you don't use the `EXCLUDE_TABLES` filter, then the job fails, because there are no tables included in a table-model export. However, because we only want to export the view `SCOTT.EMPV` as if it was a table, and we specifically do not want to export the table `SCOTT.EMP`, we set `EXCLUDE_TABLES`. The tables in the source are excluded, and the view `SCOTT.EMPV` is exported to the target as a table.

METADATA_REMAP Procedure

This procedure specifies a remapping to be applied to objects as they are processed in the specified job.

Syntax

```
DBMS_DATAPUMP.METADATA_REMAP (
   handle      IN NUMBER,
   name        IN VARCHAR2,
   old_value   IN VARCHAR2,
   value       IN VARCHAR2,
   object_type IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 64-16   METADATA_REMAP Procedure Parameters

Parameter	Description
`handle`	The handle for the current job. The current session must have previously attached to the handle through a call to the `OPEN` function.
`name`	The name of the remap. See Table 64-17 for descriptions of the available remaps.
`old_value`	Specifies which value in the dump file set should be reset to `value`
`value`	The value of the parameter for the remap. This signifies the new value that `old_value` should be translated into.
`object_type`	Designates the object type to which the remap applies. The list of object types supported for each mode are contained in the `DATABASE_EXPORT_OBJECTS`, `SCHEMA_EXPORT_OBJECTS`, `TABLE_EXPORT_OBJECTS`, and `TABLESPACE_EXPORT_OBJECTS` catalog views. By default, the remap applies to all applicable objects within the job. The `object_type` parameter allows a caller to specify different parameters for different object types within a job. Remaps that explicitly specify an object type override remaps that apply to all object types.

Table 64-17 describes the remaps provided by the `METADATA_REMAP` procedure.

Table 64-17   Remaps Provided by the METADATA_REMAP Procedure

Name	Datatype	Object Type	Meaning
REMAP_SCHEMA	TEXT	Schema objects	Any schema object in the job that matches the object_type parameter and was located in the old_value schema will be moved to the value schema. Privileged users can perform unrestricted schema remaps. Nonprivileged users can perform schema remaps only if their schema is the target schema of the remap. For example, SCOTT can remap his BLAKE's objects to SCOTT, but SCOTT cannot remap SCOTT's objects to BLAKE.
REMAP_TABLESPACE	TEXT	TABLE, INDEX, ROLLBACK_SEGMENT, MATERIALIZED_VIEW, MATERIALIZED_VIEW_LOG,TABLE_SPACE	Any storage segment in the job that matches the object_type parameter and was located in the old_value tablespace will be relocated to the value tablespace.
REMAP_DATAFILE	TEXT	LIBRARY, TABLESPACE, DIRECTORY	If old_value and value are both full file specifications, then any data file reference in the job that matches the object_type parameter and that referenced the old_value data file will be redefined to use the value data file. If old_value and value are both directory paths, then any data file reference whose object path matches old_value will have its path substituted with value.
REMAP_TABLE	TEXT	TABLE	Any reference to a table in the job that matches the old_value table name will be replaced with the value table name. The old_value parameter may refer to a partition such as employees.low. This allows names for tables constructed the by PARTITION_OPTIONS=DEPARTITION parameter to be specified by the user.

Exceptions

- INVALID_HANDLE. The specified handle is not attached to a Data Pump job.
- INVALID_ARGVAL. This message can indicate any of the following:

  - The job's mode does not include the specified `object_type`.
  - The remap has already been specified for the specified `old_value` and `object_type`.
- `INVALID_OPERATION`. Remaps are only supported for SQL_FILE and Import operations. The job's operation was Export, which does not support the use of metadata remaps.
- `INVALID_STATE`. The user called `METADATA_REMAP` after the job had started (that is, the job was not in the defining state).
- `INCONSISTENT_ARGS`. There was no `value` supplied or it was of the wrong datatype for the remap.
- `PRIVILEGE_ERROR`. A nonprivileged user attempted to do a `REMAP_SCHEMA` to a different user's schema or a `REMAP_DATAFILE`.
- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.
- `NO_SUCH_JOB`. The specified job does not exist.

Usage Notes

- The `METADATA_REMAP` procedure is only supported for Import and SQL_FILE operations. It enables you to apply commonly desired, predefined remappings to the definition of objects as part of the transfer. If you need remaps that are not supported within this procedure, you should do a preliminary SQL_FILE operation to produce a SQL script corresponding to the dump file set. By editing the DDL directly and then executing it, you can produce any remappings that you need.
- Transforms for the DataPump API are a subset of the remaps implemented by the `DBMS_METADATA.SET_TRANSFORM_PARAMETER` API. Multiple remaps can be defined for a single job. However, each remap defined must be unique according its parameters. That is, two remaps cannot specify conflicting or redundant remaps.

METADATA_TRANSFORM Procedure

This procedure specifies transformations to be applied to objects as they are processed in the specified job.

Syntax

```
DBMS_DATAPUMP.METADATA_TRANSFORM (
   handle      IN NUMBER,
   name        IN VARCHAR2,
   value       IN VARCHAR2,
   object_type IN VARCHAR2 DEFAULT NULL);

DBMS_DATAPUMP.METADATA_TRANSFORM (
   handle      IN NUMBER,
   name        IN VARCHAR2,
   value       IN NUMBER,
   object_type IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 64-18 METADATA_TRANSFORM Procedure Parameters

Parameter	Description
`handle`	The handle for the current job. The current session must have previously attached to the handle through a call to the `OPEN` function.
`name`	The name of the transformation.
`value`	The value of the parameter for the transform
`object_type`	Designates the object type to which the transform applies. The list of object types supported for each mode are contained in the `DATABASE_EXPORT_OBJECTS`, `SCHEMA_EXPORT_OBJECTS`, `TABLE_EXPORT_OBJECTS`, and `TABLESPACE_EXPORT_OBJECTS` catalog views. By default, the transform applies to all applicable objects within the job. The `object_type` parameter allows a caller to specify different transform parameters for different object types within a job. Transforms that explicitly specify an object type override transforms that apply to all object types.

The following table describes the transforms provided by the `METADATA_TRANSFORM` procedure.

Table 64-19 Transforms Provided by the METADATA_TRANFORM Procedure

Name	Datatype	Object Type	Meaning
`DISABLE_ARCHIVE_LOGGING`	`NUMBER`	`TABLE` `INDEX`	Specifies whether to disable archive logging for specified object types during import. A value of zero (`FALSE`) is the default. It specifies that archive logging will take place. This is the default behavior if this transform parameter is not specified. A non-zero (`TRUE`) value disables archive logging for the specified object types before data is loaded during import. If no object type is specified, then archive logging is disabled for both TABLE and INDEX object types. All other object types processed during data pump import are logged. **Note:** If the database is in `FORCE LOGGING` mode, then the `DISABLE_ARCHIVE_LOGGING` transform does not disable logging when indexes and tables are created.

Table 64-19 (Cont.) Transforms Provided by the METADATA_TRANFORM Procedure

Name	Datatype	Object Type	Meaning
INMEMORY	NUMBER	TABLE TABLESPACE MATERIALIZED VIEWS	The INMEMORY transform is related to the In-Memory Column Store (IM column store), an optional area in the SGA that stores whole tables, table partitions, and individual columns in a compressed columnar format. If a non-zero value (TRUE) is specified on import, then Data Pump generates an IM clause that preserves the IM setting those objects had at export time. This is the default. If a value of zero (FALSE) is specified on import, then Data Pump does not include an IM clause for any objects. **Note**: The INMEMORY transform is available only in Oracle Database 12c Release 1 (12.1.0.2) or higher.
INMEMORY_CLAUSE	TEXT	TABLE TABLESPACE MATERIALIZED VIEWS	The INMEMORY_CLAUSE transform is related to the In-Memory Column Store (IM column store), an optional area in the SGA that stores whole tables, table partitions, and individual columns in a compressed columnar format. When you specify this transform, Data Pump uses the contents of the string as the IM clause for all objects being imported that have an IM clause in their DDL. This transform is useful when you want to override the IM clause for an object in the dump file. **Note**: The INMEMORY_CLAUSE transform is available only in Oracle Database 12c Release (12.1.0.2) or higher.
LOB_STORAGE	TEXT	TABLE	Specifies the storage type to use for LOB segments. The options are as follows: • SECUREFILE - LOB storage is returned as SECUREFILE • BASICFILE - LOB storage is returned as BASICFILE • DEFAULT - The keyword (SECUREFILE or BASICFILE) is omitted in the LOB STORE AS clause. • NO_CHANGE - LOB segments are created with the same storage they had in the source database. This is the default. Specifying this transform changes the LOB storage for all tables in the job, including tables that provide storage for materialized views.
OID	NUMBER	TYPE TABLE	If zero, inhibits the assignment of the exported OID during type or table creation. Instead, a new OID will be assigned. Use of this transform on Object Tables will cause breakage in REF columns that point to the table. Defaults to 1.

Table 64-19 (Cont.) Transforms Provided by the METADATA_TRANFORM Procedure

Name	Datatype	Object Type	Meaning
OMIT_ACDR_METADATA	NUMBER	TABLE	Used with Oracle GoldenGate. When set to 1 (true), excludes invisible columns from importing replicated tables, deletes tombstone tables, and deletes all the automatic conflict detection and resolution (ACDR) instance procedural actions.
PCTSPACE	NUMBER	TABLE INDEX TABLESPACE	Specifies a percentage multiplier used to alter extent allocations and data file sizes. Used to shrink large tablespaces for testing purposes. Defaults to 100.
SEGMENT_ATTRIBUTES	NUMBER	TABLE, INDEX	If nonzero (TRUE), emit storage segment parameters. Defaults to 1.
SEGMENT_CREATION	NUMBER	TABLE	If nonzero (TRUE), the SQL SEGMENT CREATION clause is added to the CREATE TABLE statement. That is, the CREATE TABLE statement will explicitly say either SEGMENT CREATION DEFERRED or SEGMENT CREATION IMMEDIATE. If the value is FALSE, then the SEGMENT CREATION clause is omitted from the CREATE TABLE statement. Set this parameter to FALSE to use the default segment creation attributes for the table(s) being loaded. Defaults to nonzero (TRUE).
STORAGE	NUMBER	TABLE	If nonzero (TRUE), emit storage clause. (Ignored if SEGMENT_ATTRIBUTES is zero.) Defaults to nonzero (TRUE).
TABLE_COMPRESSION_CLAUSE	TEXT	TABLE	Specifies a table compression clause (for example, COMPRESS BASIC) to use when the table is created. Specify NONE to omit the table compression clause. The table will have the default compression for the tablespace. Specifying this transform changes the compression type for all tables in the job, including tables that provide storage for materialized views.

Table 64-19 (Cont.) Transforms Provided by the METADATA_TRANFORM Procedure

Name	Datatype	Object Type	Meaning
`XMLTYPE_STORAGE_CLAUSE`	`TEXT`	`TABLE`	Specifies to use the `TRANSPORTABLE BINARY XML XMLType`. This new type is the recommended storage type for Oracle Database 23ai to store the data in a self-contained binary format or the non-transportable compact BINARY XML storage XMLType for XML data storage. There is no default. If the transform is not used, then the source datatype in the dumpfile is the datatype defined on the target. syntax: dbms_datapump.metadata_transform (handle => h1, name => 'XMLTYPE_STORAGE_CLAUSE', value => 'transform param value');

Exceptions

- `INVALID_HANDLE`. The specified handle is not attached to a Data Pump job.
- `INVALID_ARGVAL`. This message can indicate any of the following:
  - The mode is transportable, which doesn't support transforms.
  - The job's mode does not include the specified `object_type`.
  - The transform has already been specified for the specified `value` and `object_type`.
- `INVALID_OPERATION`. Transforms are only supported for SQL_FILE and Import operations. The job's operation was Export which does not support the use of metadata transforms.
- `INVALID_STATE`. The user called `METADATA_TRANSFORM` after the job had started (that is, the job was not in the defining state).
- `INCONSISTENT_ARGS`. There was no `value` supplied or it was of the wrong datatype for the transform.
- `PRIVILEGE_ERROR`. A nonprivileged user attempted to do a `REMAP_SCHEMA` to a different user's schema or a `REMAP_DATAFILE`.
- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.
- `NO_SUCH_JOB`. The specified job does not exist.

Usage Notes

- The `METADATA_TRANSFORM` procedure is only supported for Import and SQL_FILE operations. It enables you to apply commonly desired, predefined transformations to the definition of objects as part of the transfer. If you need transforms that are not supported within this procedure, you should do a preliminary SQL_FILE operation to produce a SQL script corresponding to the dump file set. By editing the DDL directly and then executing it, you can produce any transformations that you need.
- Transforms for the Oracle Data Pump API are a subset of the transforms implemented by the `DBMS_METADATA.SET_TRANSFORM_PARAMETER` API. Multiple transforms can be defined for

a single job. However, each transform defined must be unique according its parameters. That is, two transforms cannot specify conflicting or redundant transformations.

- In Oracle Database 23ai, Oracle recommends that you use the `TRANSPORTABLE BINARY XML XMLType` to store the data in a self-contained binary format. In Oracle Database 19c and 21c, you can use this XMLType in Cloud deployments. This XMLType does not store the metadata used to encode or decode XML data in a central table, which simplifies the XML data storage and makes it easier to transport. For more information about this XMLType, see *Oracle XML DB Developer's Guide*.

Related Topics

- Export/Import Limitations for Oracle XML DB Repository in *Oracle XML DB Developer's Guide*

OPEN Function

This function is used to declare a new job using the Data Pump API.

Note:

A multitenant container database is the only supported architecture in Oracle Database 21c and later releases. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

The handle that is returned is used as a parameter for calls to all other procedures (but not to the `ATTACH` function).

Syntax

```
DBMS_DATAPUMP.OPEN (
   operation    IN VARCHAR2,
   job_mode         IN VARCHAR2,
   remote_link  IN VARCHAR2 DEFAULT NULL,
   job_name     IN VARCHAR2 DEFAULT NULL,
   version      IN VARCHAR2 DEFAULT 'COMPATIBLE')
 RETURN NUMBER;
```

Parameters

Table 64-20 OPEN Function Parameters

Parameter	Meaning
`operation`	The type of operation to be performed. Table 64-21 contains descriptions of valid operation types.
`job_mode`	The scope of the operation to be performed. Table 64-22 contains descriptions of valid modes. Specifying NULL generates an error.
`remote_link`	If the value of this parameter is non-null, it provides the name of a database link to the remote database that will be the source of data and metadata for the current job.

Table 64-20 (Cont.) OPEN Function Parameters

Parameter	Meaning
`job_name`	The name of the job. The name is limited to 128 characters; it will be truncated if more than 128 characters are used. It may consist of printable characters and spaces. It is implicitly qualified by the schema of the user executing the `OPEN` function and must be unique to that schema (that is, there cannot be other Data Pump jobs using the same name). The name is used to identify the job both within the API and with other database components such as identifying the job in the `DBA_RESUMABLE` view if the job becomes suspended through lack of resources. If no name is supplied, a system generated name will be provided for the job in the following format: `"SYS_<OPERATION>_<MODE>_%N"`. The default job name is formed where `%N` expands to a two-digit incrementing integer starting at '`01`' (for example, "`SYS_IMPORT_FULL_03`"). The name supplied for the job will also be used to name the master table and other resources associated with the job.
`version`	The version of database objects to be extracted. This option is only valid for Export, network Import, and SQL_FILE operations. Database objects or attributes that are incompatible with the version will not be extracted. Legal values for this parameter are as follows: • `COMPATIBLE` - (default) the version of the metadata corresponds to the database compatibility level and the compatibility release level for feature (as given in the `V$COMPATIBILITY` view). Database compatibility must be set to 9.2 or higher. • `LATEST` - the version of the metadata corresponds to the database version. • A specific database version, for example, '11.0.0'. Specify a value of 12 to allow all existing database features, components, and options to be exported from Oracle Database 11*g* release 2 (11.2.0.3) or later into an Oracle Database 12 *c* Release 1 (12.1) (either a multitenant container database (CDB) or a non-CDB).

Table 64-21 describes the valid operation types for the `OPEN` function.

Table 64-21 Valid Operation Types for the OPEN Function

Operation	Description
`EXPORT`	Saves data and metadata to a dump file set or obtains an estimate of the size of the data for an operation.
`IMPORT`	Restores data and metadata from a dump file set or across a database link.
`SQL_FILE`	Displays the metadata within a dump file set, or from across a network link, as a SQL script. The location of the SQL script is specified through the `ADD_FILE` procedure.

Table 64-22 describes the valid modes for the `OPEN` function.

Table 64-22 Valid Modes for the OPEN Function

Mode	Description
FULL	Operates on the full database or full dump file set except for Oracle Database internal schemas. (Some tables from Oracle Database internal schemas may be registered to be exported and imported in full operations in order to provide consistent metadata during import.) The TRANSPORTABLE parameter can be set to ALWAYS during a full database export in order to move data via transportable tablespaces rather than in the Data Pump dump file.
SCHEMA	Operates on a set of selected schemas. Defaults to the schema of the current user. All objects in the selected schemas are processed. In SCHEMA mode, you cannot specify Oracle-internal schemas (for example, SYS, XDB, ORDSYS, MDSYS, CTXSYS, ORDPLUGINS, or LBACSYS).
TABLE	Operates on a set of selected tables. Defaults to all of the tables in the current user's schema. Only tables and their dependent objects are processed.
TABLESPACE	Operates on a set of selected tablespaces. No defaulting is performed. Tables that have storage in the specified tablespaces are processed in the same manner as in Table mode.
TRANSPORTABLE	Operates on metadata for tables (and their dependent objects) within a set of selected tablespaces to perform a transportable tablespace export/import.

Return Values

- An opaque handle for the job. This handle is used as input to the following procedures: ADD_FILE, CREATE_JOB_VIEW, DATA_FILTER, DETACH, GET_STATUS, LOG_ENTRY, LOG_ERROR,METADATA_FILTER, METADATA_REMAP, METADATA_TRANSFORM, SET_PARALLEL,SET_PARAMETER, START_JOB,STOP_JOB, and WAIT_FOR_JOB

Exceptions

- INVALID_ARGVAL. An invalid operation or mode was specified. A NULL or invalid value was supplied for an input parameter. The error message text identifies the parameter.
- JOB_EXISTS. A table already exists with the specified job name.
- PRIVILEGE_ERROR. The user does not have the necessary privileges or roles to use the specified mode.
- INTERNAL_ERROR. The job was created under the wrong schema or the master table was of the wrong format.
- SUCCESS_WITH_INFO. The function succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

- When the job is created, a master table is created for the job under the caller's schema within the caller's default tablespace. A handle referencing the job is returned that attaches the current session to the job. Once attached, the handle remains valid until either an explicit or implicit detach occurs. The handle is only valid in the caller's session. Other handles can be attached to the same job from a different session by using the `ATTACH` function.

- If the call to the `OPEN` function fails, call the `GET_STATUS` procedure with a null handle to retrieve additional information about the failure.

SET_PARALLEL Procedure

This procedure adjusts the degree of parallelism within a job.

Syntax

```
DBMS_DATAPUMP.SET_PARALLEL(
   handle      IN NUMBER,
   degree      IN NUMBER);
```

Parameters

Table 64-23 SET_PARALLEL Procedure Parameters

Parameter	Description
`handle`	The handle of a job. The current session must have previously attached to the handle through a call to either the `OPEN` or `ATTACH` function.
`degree`	The maximum number of worker processes that can be used for the job. You use this parameter to adjust the amount of resources used for a job.

Exceptions

- `INVALID_HANDLE`. The specified handle is not attached to a Data Pump job.
- `INVALID_OPERATION`. The `SET_PARALLEL` procedure is only valid for export and import operations.
- `INVALID_ARGVAL`. An invalid value was supplied for an input parameter.
- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.
- `NO_SUCH_JOB`. The specified job does not exist.

Usage Notes

- The `SET_PARALLEL` procedure is only available in the Enterprise Edition of the Oracle database.
- The `SET_PARALLEL` procedure can be executed by any session attached to a job. The job must be in one of the following states: Defining, Idling, or Executing.
- The effect of decreasing the degree of parallelism may be delayed because ongoing work needs to find an orderly completion point before `SET_PARALLEL` can take effect.
- Decreasing the parallelism will not result in fewer worker processes associated with the job. It will only decrease the number of worker processes that will be executing at any given time.
- Increasing the parallelism will take effect immediately if there is work that can be performed in parallel.

- The degree of parallelism requested by a user may be decreased based upon settings in the resource manager or through limitations introduced by the PROCESSES or SESSIONS initialization parameters in the init.ora file.
- To parallelize an Export job to a degree of *n*, the user should supply *n* files in the dump file set or specify a substitution variable in a file specification. Otherwise, some of the worker processes will be idle while waiting for files.
- SQL_FILE operations always operate with a degree of 1. Jobs running in the Transportable mode always operate with a degree of 1.

SET_PARAMETER Procedures

The DBMS_DATAPUMP procedure SET_PARAMETER is used to specify job-processing options.

Syntax

```
DBMS_DATAPUMP.SET_PARAMETER(
   handle       IN NUMBER,
   name         IN VARCHAR2,
   value        IN VARCHAR2);

DBMS_DATAPUMP.SET_PARAMETER (
   handle      IN NUMBER,
   name        IN VARCHAR2,
   value       IN NUMBER);
```

Parameters

Table 64-24 SET_PARAMETER Procedure Parameters

Parameter	Description
handle	The handle of a job. The current session must have previously attached to the handle through a call to the OPEN function.
name	The name of the parameter.
value	The value for the specified parameter

The following table describes the valid options for the name parameter of the SET_PARAMETER procedure.

Table 64-25 Valid Options for the name Parameter in the SET_PARAMETER Procedure

Parameter Name	Datatype	Supported Operations	Meaning
CLIENT_COMMAND	TEXT	All	An opaque string used to describe the current operation from the client's perspective. The command-line procedures will use this string to store the original command used to invoke the job.

Table 64-25 (Cont.) Valid Options for the name Parameter in the SET_PARAMETER Procedure

Parameter Name	Datatype	Supported Operations	Meaning
COMPRESSION	TEXT	Export	Allows you to trade off the size of the dump file set versus the time it takes to perform export and import operations. The DATA_ONLY option compresses only user data in the dump file set. The METADATA_ONLY option compresses only metadata in the dump file set. The ALL option compresses both user data and metadata. The NONE option stores the dump file set in an uncompressed format. The METADATA_ONLY and NONE options require a job version of 10.2 or later. All other options require a job version of 11.1 or later. Default=METADATA_ONLY
COMPRESSION_ALGORITHM	TEXT	Export	Indicates the compression algorithm is to be used when compressing dump file data. The choices are as follows: • BASIC--Offers a good combination of compression ratios and speed; the algorithm used is the same as in previous versions of Oracle Data Pump. • LOW---Least impact on backup throughput and suited for environments where CPU resources are the limiting factor. • MEDIUM---Recommended for most environments. This option, like the BASIC option, provides a good combination of compression ratios and speed, but it uses a different algorithm than BASIC. • HIGH--Best suited for exports over slower networks where the limiting factor is network speed. To use this feature, the COMPATIBLE initialization parameter must be set to at least 12.0.0. This feature requires that the Oracle Advanced Compression option is enabled.
DATA_ACCESS_METHOD	TEXT	Export and Import	Allows you to specify an alternative method for Oracle Data Pump Export (unloading) or Oracle Data Pump Import (loading) to unload or load data if the default method does not work for some reason. For Export, the options are AUTOMATIC, DIRECT_PATH, or EXTERNAL_TABLE. For Import, the options are AUTOMATIC, DIRECT_PATH, EXTERNAL_TABLE, CONVENTIONAL, or (network imports only) INSERT_AS_SELECT. The INSERT_AS_SELECT option is valid only if you are importing directly over a database link. If you are importing directly over a database link, then the allowed options are AUTOMATIC, DIRECT_PATH, and INSERT_AS_SELECT. **Note**: For both Export and Import, Oracle recommends that you use the default option (AUTOMATIC) whenever possible because it allows Oracle Data Pump to automatically select the most efficient method.

Table 64-25 (Cont.) Valid Options for the name Parameter in the SET_PARAMETER Procedure

Parameter Name	Datatype	Supported Operations	Meaning

`DATA_OPTIONS`	Number	Export and Import	A bitmask to supply special options for processing the job. The possible values are as follows:
			`KU$_DATAOPT_SKIP_CONST_ERR`
			`KU$_DATAOPT_XMLTYPE_CLOB`
			`KU$_DATAOPT_DISABL_APPEND_HINT`
			`KU$_DATAOPT_GRP_PART_TAB`
			`KU$_DATAOPT_TRUST_EXIST_TB_PAR`
			`KU$_DATAOPT_CONT_LOAD_ON_FMT_ERR`
			`KU$_DATAOPT_REJECT_ROWS_REPCHR`
			`KU$_DATAOPT_ENABLE_NET_COMP`
			`KU$_DATAOPT_VALIDATE_TBL_DATA`
			`KU$_DATAOPT_VERIFY_STREAM_FORM`
			Export supports the value `KU$_DATAOPT_XMLTYPE_CLOB`. This option stores compressed `XMLType` columns in the dump file in `CLOB` format rather than as XML documents.
			Note: `XMLType` stored as `CLOB` is deprecated as of Oracle Database 12c Release 1 (12.1).
			Import supports the value `KU$_DATAOPT_SKIP_CONST_ERR`. This option specifies that if constraint violations occur while data is being imported into user tables, the rows that cause the violations will be rejected and the load will continue. If this option is not set, a constraint error will abort the loading of the entire partition (or table for unpartitioned tables). Setting this option may affect performance, especially for pre-existing tables with unique indexes or constraints.
			Import also supports the value `KU$_DATAOPT_DISABL_APPEND_HINT`. This option prevents the append hint from being applied to the data load. Disabling the `APPEND` hint can be useful if there is a small set of data objects to load that already exist in the database and some other application may be concurrently accessing one or more of the data objects.
			Oracle Data Pump Export supports the value `KU$_DATAOPT_GRP_PART_TAB`. This option tells Data Pump to unload all table data in one operation rather than unload each table partition as a separate operation. As a result, the definition of the table will not matter at import time because Import will see one partition of data that will be loaded into the entire table.
			Oracle Data Pump Import supports the value `KU$_DATAOPT_TRUST_EXIST_TB_PAR`. This option tells Data Pump to load partition data
			in parallel into existing tables.
			Use of the `DATA_OPTIONS` parameter requires that the `version` on the `OPEN` function be set to 11.1 or later.
			Default=0
			Oracle Data Pump Import supports the value `KU$_DATAOPT_CONT_LOAD_ON_FMT_ERR`. This option tells Oracle Data Pump to skip forward to the start of the next granule if a stream format error is encountered while loading table data. Most stream format errors are caused by corrupt dump files. This value can be used if Oracle Data Pump encounters a stream format error and the original export database is not available to export the table data

Table 64-25 (Cont.) Valid Options for the name Parameter in the SET_PARAMETER Procedure

Parameter Name	Datatype	Supported Operations	Meaning
			again. If Oracle Data Pump skips over data, not all data from the source database are imported, which potentially can mean skipping hundreds or thousands of rows.
`ENCRYPTION`	`TEXT`	Export	Specifies what to encrypt in the dump file set, as follows: `ALL` enables encryption for all data and metadata in the export operation. `DATA_ONLY` specifies that only data is written to the dump file set in encrypted format. `ENCRYPTED_COLUMNS_ONLY` specifies that only encrypted columns are written to the dump file set in encrypted format. `METADATA_ONLY` specifies that only metadata is written to the dump file set in encrypted format. `NONE` specifies that no data is written to the dump file set in encrypted format. This parameter requires a job version of 11.1 or later. The default value depends upon the combination of encryption-related parameters that are used. To enable encryption, either `ENCRYPTION` or `ENCRYPTION_PASSWORD` or both, must be specified. If only `ENCRYPTION_PASSWORD` is specified, then `ENCRYPTION` defaults to `ALL`. If neither `ENCRYPTION` nor `ENCRYPTION_PASSWORD` is specified, then `ENCRYPTION` defaults to `NONE`. To specify `ALL`, `DATA_ONLY`, or `METADATA_ONLY`, the `COMPATIBLE` initialization parameter must be set to at least 11.1. **NOTE**: If the data being exported includes SecureFiles that you want to be encrypted, then you must specify `ENCRYPTION=ALL` to encrypt the entire dump file set. Encryption of the entire dump file set is the only way to achieve encryption security for SecureFiles during an Oracle Data Pump export operation.
`ENCRYPTION_ALGORITHM`	`TEXT`	Export	Identifies which cryptographic algorithm should be used to perform encryption. Possible values are `AES128`, `AES192`, and `AES256`. The `ENCRYPTION_ALGORITHM` parameter requires that you also specify either `ENCRYPTION` or `ENCRYPTION_PASSWORD`; otherwise an error is returned. For the most current information, see: Supported Encryption and Integrity Algorithms in *Oracle Database Transparent Data Encryption Guide*. This parameter requires a job version of 1.1 or later. Default=`AES128`

Table 64-25 (Cont.) Valid Options for the name Parameter in the SET_PARAMETER Procedure

Parameter Name	Datatype	Supported Operations	Meaning
ENCRYPTION_MODE	TEXT	Export	Identifies the types of security used for encryption and decryption. The values are as follows: PASSWORD requires that you provide a password when creating encrypted dump file sets. You will need to provide the same password when you import the dump file set. PASSWORD mode requires that you also specify the ENCRYPTION_PASSWORD parameter. The PASSWORD mode is best suited for cases in which the dump file set will be imported into a different or remote database, but which must remain secure in transit. TRANSPARENT allows an encrypted dump file set to be created without any intervention from a database administrator (DBA), provided the required Oracle Encryption Wallet is available. Therefore, the ENCRYPTION_PASSWORD parameter is not required, and will in fact, cause an error if it is used in TRANSPARENT mode. This encryption mode is best suited for cases in which the dump file set will be imported into the same database from which it was exported. DUAL creates a dump file set that can later be imported using either the Oracle Encryption Wallet or the password that was specified with the ENCRYPTION_PASSWORD parameter. DUAL mode is best suited for cases in which the dump file set will be imported on-site using the Oracle Encryption Wallet, but which may also need to be imported offsite where the Oracle Encryption Wallet is not available. When you use the ENCRYPTION_MODE parameter, you must also use either the ENCRYPTION or ENCRYPTION_PASSWORD parameter. Otherwise, an error is returned. To use DUAL or TRANSPARENT mode, the COMPATIBLE initialization parameter must be set to at least 11.1. The default mode depends on which other encryption-related parameters are used. If only ENCRYPTION is specified, then the default mode is TRANSPARENT. If ENCRYPTION_PASSWORD is specified and the Oracle Encryption Wallet is open, then the default is DUAL. If ENCRYPTION_PASSWORD is specified and the Oracle Encryption Wallet is closed, then the default is PASSWORD.
ENCRYPTION_PASSWORD	TEXT	Export and Import	For export operations, this parameter is required if ENCRYPTION_MODE is set to either PASSWORD or DUAL. It is also required for transportable export/import operations (job mode=FULL and TRANSPORTABLE=ALWAYS) when the database includes either encrypted tablespaces or tables with encrypted columns.
ESTIMATE	TEXT	Export and Import	Specifies that the estimate method for the size of the tables should be performed before starting the job. If BLOCKS, a size estimate for the user tables is calculated using the count of blocks allocated to the user tables. If STATISTICS, a size estimate for the user tables is calculated using the statistics associated with each table. If no statistics are available for a table, the size of the table is estimated using BLOCKS. The ESTIMATE parameter cannot be used in Transportable Tablespace mode. Default=STATISTICS

Table 64-25 (Cont.) Valid Options for the name Parameter in the SET_PARAMETER Procedure

Parameter Name	Datatype	Supported Operations	Meaning
`ESTIMATE_ONLY`	`NUMBER`	Export	Specifies that only the estimation portion of an export job should be performed. This option is useful for estimating the size of dump files when the size of the export is unknown.
`FLASHBACK_SCN`	`NUMBER`	Export and network Import	System change number (SCN) to serve as transactionally consistent point for reading user data. If neither `FLASHBACK_SCN` nor `FLASHBACK_TIME` is specified, there will be no transactional consistency between partitions, except for logical standby databases and Streams targets. `FLASHBACK_SCN` is not supported in Transportable mode. For FLASHBACK_SCN, Oracle Data Pump supports the 8–byte big SCNs used in Oracle Database 12c release 2 (12.2) and later releases.
`FLASHBACK_TIME`	`TEXT`	Export and network Import	Either the date and time used to determine a consistent point for reading user data or a string of the form `TO_TIMESTAMP(...)`. If neither `FLASHBACK_SCN` nor `FLASHBACK_TIME` is specified, there will be no transactional consistency between partitions. `FLASHBACK_SCN` and `FLASHBACK_TIME` cannot both be specified for the same job. `FLASHBACK_TIME` is not supported in Transportable mode.
`INCLUDE_METADATA`	`NUMBER`	Export and Import	If nonzero, then metadata for objects will be moved in addition to user table data. If zero, then metadata for objects will not moved. This parameter converts an Export operation into an unload of user data and an Import operation into a load of user data. `INCLUDE_METADATA` is not supported in Transportable mode. Default=1
`INDEX_THRESHOLD`	TEXT	Import	Sets a size threshold for creating large indexes with a degree of parallelism (DOP) greater than 1 in conjunction with the `ONESTEP_INDEX` parameter. Indexes equal to and above the threshold can be created with a degre of parallism DOP greater than 1 if the parameter `ONESTEP_INDEX=FALSE`. The `INDEX_THRESHOLD` value must be specified as a text string, such as 1000B, 100k, 200kb, 100M, 200mb, 100G, 200gb, 100t, 200TB. An invalid threshold will produce an error. Oracle recommends that you use the default in most cases. Default=150M
`KEEP_MASTER`	`NUMBER`	Export and Import	Specifies whether the master table should be deleted or retained at the end of a Data Pump job that completes successfully. The master table is automatically retained for jobs that do not complete successfully. Default=0
`LOGTIME`	`TEXT`	Export and Import	Specifies that messages displayed during export and import operations be timestamped. Valid options are as follows: • `NONE`--No timestamps on status or log file messages (this is the default) • `STATUS`--Timestamps on status messages only • `LOGFILE`--Timestamps on log file messages only • `ALL`--Timestamps on both status and log file messages

Table 64-25 (Cont.) Valid Options for the name Parameter in the SET_PARAMETER Procedure

Parameter Name	Datatype	Supported Operations	Meaning
`MASTER_ONLY`	`NUMBER`	Import	Indicates whether to import just the master table and then stop the job so that the contents of the master table can be examined. Default=0
`METRICS`	`NUMBER`	Export and Import	Indicates whether additional information about the job should be reported to the Data Pump log file. Default=0
`ONESTEP_INDEX`	`NUMBER`	Import	Attempts to optimize index creation concurrency and balance it with job parallelism in conjunction with the `INDEX_THRESHOLD` parameter. After a method is selected, it cannot be changed for the job, including a job restart. Valid options: 0 (False), and 1 (True). In most cases, 0 will yield better performance than 1: `ONESTEP_INDEX=0`, in conjunction with the `INDEX_THRESHOLD` parameter starts a two-step process to balance parallelism for creating very large indexes with parallelism for importing all objects in the overall import job, up to the `PARALLEL` setting for the job. Oracle Data Pump attempts to create all the indexes in the shortest amount of time, given the constraints of the job. Large index creation is prioritized first because they take longer to create. `ONESTEP_INDEX=1` uses a single step process and a DOP of 1 for each index created during import. However, multiple indexes can be created in parallel up to the DOP setting for the import job. 1 (true) can be optimal if all indexes are relatively small, below the `INDEX_THRESHOLD` default value. 1 is also the default for jobs with `PARALLEL=1`.
`PARTITION_OPTIONS`	`TEXT`	Import	Specifies how partitioned tables should be handled during an import operation. The options are as follows: `NONE` means that partitioning is reproduced on the target database as it existed in the source database. `DEPARTITION` means that each partition or subpartition that contains storage in the job is reproduced as a separate unpartitioned table. Intermediate partitions that are subpartitioned are not re-created (although their subpartitions are converted into tables). The names of the resulting tables are system-generated from the original table names and partition names unless the name is overridden by the `REMAP_TABLE` metadata transform. `MERGE` means that each partitioned table is re-created in the target database as an unpartitioned table. The data from all of the source partitions is merged into a single storage segment. This option is not supported for transportable jobs or when the `TRANSPORTABLE` parameter is set to `ALWAYS`. This parameter requires a job version of 11.1 or later. Default=`NONE`
`REUSE_DATAFILES`	`NUMBER`	Import	Specifies whether the import job should reuse existing data files for tablespace creation. Default=0
`SKIP_UNUSABLE_INDEXES`	`NUMBER`	Import	If nonzero, rows will be inserted into tables having unusable indexes. `SKIP_UNUSABLE_INDEXES` is not supported in Transportable mode. Default=1

Table 64-25 (Cont.) Valid Options for the name Parameter in the SET_PARAMETER Procedure

Parameter Name	Datatype	Supported Operations	Meaning
`SOURCE_EDITION`	`TEXT`	Export and network Import	The application edition that will be used for determining the objects that will be unloaded for export and for network import.
`STREAMS_CONFIGURATION`	`NUMBER`	Import	Specifies whether to import any Streams metadata that may be present in the export dump file. Default=1
`TABLE_EXISTS_ACTION`	`TEXT`	Import	Specifies the action to be performed when data is loaded into a preexisting table. The possible actions are: `TRUNCATE`, `REPLACE`, `APPEND`, and `SKIP`. If `INCLUDE_METADATA=0`, only `TRUNCATE` and `APPEND` are supported. If `TRUNCATE`, rows are removed from a preexisting table before inserting rows from the Import. Note that if `TRUNCATE` is specified on tables referenced by foreign key constraints, the `TRUNCATE` will be modified into a `REPLACE`. If `REPLACE`, preexisting tables are replaced with new definitions. Before creating the new table, the old table is dropped. If `APPEND`, new rows are added to the existing rows in the table. If `SKIP`, the preexisting table is left unchanged. `TABLE_EXISTS_ACTION` is not supported in Transportable mode. The default is `SKIP` if metadata is included in the import. The default is `APPEND` if `INCLUDE_METADATA` is set to 0.
`TABLESPACE_DATAFILE`	`TEXT`	Import	Specifies the full file specification for a data file in the transportable tablespace set. `TABLESPACE_DATAFILE` is only valid for transportable mode imports. `TABLESPACE_DATAFILE` can be specified multiple times, but the value specified for each occurrence must be different.
`TARGET_EDITION`	`TEXT`	Import	The application edition that will be used for determining where the objects will be loaded for import and for network import.

Table 64-25 (Cont.) Valid Options for the name Parameter in the SET_PARAMETER Procedure

Parameter Name	Datatype	Supported Operations	Meaning
`TRANSPORTABLE`	`TEXT`	Export (and network import or full-mode import)	This option is for export operations done in table mode, and also for full-mode imports and network imports. It allows the data to be moved using transportable tablespaces. In table-mode storage segments in the moved tablespaces that are not associated with the parent schemas (tables) will be reclaimed at import time. If individual partitions are selected in a table-mode job, only the tablespaces referenced by those partitions will be moved. During import, the moved partitions can only be reconstituted as tables by using the `PARTITION_OPTIONS=DEPARTITION` parameter. Use of the `TRANSPORTABLE` parameter prohibits the subsequent import of the dump file into a database at a lower version or using different character sets. Additionally, the data files may need to be converted if the target database is on a different platform. In table-mode, the `TRANSPORTABLE` parameter is not allowed if a network link is supplied on the `OPEN` call. The possible values for this parameter are as follows: `ALWAYS` - data is always moved by moving data files. This option is valid only for table mode and full mode. `NEVER` - data files are never used for copying user data This parameter requires a job version of 11.1 or later This parameter requires a job version of 12.1 or later when the job mode is FULL. Default=`NEVER`
`TTS_FULL_CHECK`	`NUMBER`	Export	If nonzero, verifies that a transportable tablespace set has no dependencies (specifically, IN pointers) on objects outside the set, and vice-versa. Only valid for Transportable mode Exports. Default=`0`
`USER_METADATA`	`NUMBER`	Export and network Import	For schema-mode operations, specifies that the metadata to re-create the users' schemas (for example, privilege grants to the exported schemas) should also be part of the operation if set to nonzero. Users must be privileged to explicitly set this parameter. The `USER_METADATA` parameter cannot be used in Table, Tablespace, or Transportable Tablespace mode. Default=`1` if user has `DATAPUMP_EXP_FULL_DATABASE` role; 0 otherwise.

Exceptions

- `INVALID_HANDLE`. The specified handle is not attached to a Data Pump job.
- `INVALID_ARGVAL`. This exception could be due to any of the following causes:
  - An invalid name was supplied for an input parameter
  - The wrong datatype was used for `value`
  - A `value` was not supplied
  - The supplied `value` was not allowed for the specified parameter name
  - A flashback parameter had been established after a different flashback parameter had already been established

  – A parameter was specified that did not support duplicate definitions

- INVALID_OPERATION. The operation specified is invalid in this context.
- INVALID_STATE. The specified job is not in the Defining state.
- INCONSISTENT_ARGS. Either the specified parameter is not supported for the current operation type or it is not supported for the current mode.
- PRIVILEGE_ERROR. The user does not have the DATAPUMP_EXP_FULL_DATABASE or DATAPUMP_IMP_FULL_DATABASE role required for the specified parameter.
- SUCCESS_WITH_INFO. The procedure succeeded, but further information is available through the GET_STATUS procedure.
- NO_SUCH_JOB. The specified job does not exist.

Usage Notes

- The SET_PARAMETER procedure is used to specify optional features for the current job. For a list of supported options, see the preceding table, "Valid Options for the name Parameter in the SET_PARAMETER Procedure".

START_JOB Procedure

This procedure begins or resumes job execution.

Syntax

```
DBMS_DATAPUMP.START_JOB (
   handle        IN NUMBER,
   skip_current     IN  NUMBER DEFAULT 0,
   abort_step       IN  NUMBER DEFAULT 0,
   cluster_ok       IN  NUMBER DEFAULT 1,
   service_name     IN  VARCHAR2 DEFAULT NULL);
```

Parameters

Table 64-26 START_JOB Procedure Parameters

Parameter	Description
handle	The handle of a job. The current session must have previously attached to the handle through a call to either the OPEN or ATTACH function.
skip_current	If nonzero, causes actions that were 'in progress' on a previous execution of the job to be skipped when the job restarts. The skip will only be honored for Import jobs. This mechanism allows the user to skip actions that trigger fatal bugs and cause the premature termination of a job. Multiple actions can be skipped on a restart. The log file will identify which actions are skipped. If a domain index was being processed, all pieces of the domain index are skipped even if the error occurred in only a subcomponent of the domain index. A description of the actions skipped is entered into the log file. skip_current is ignored for the initial START_JOB in a job. If zero, no data or metadata is lost upon a restart.
abort_step	Value must be 0. Inserting values other than 0 into this argument will have unintended consequences.
cluster_ok	If = 0, all workers are started on the current instance. Otherwise, workers are started on instances usable by the job.

Table 64-26 (Cont.) START_JOB Procedure Parameters

Parameter	Description
`service_name`	If specified, indicates a service name used to constrain the job to specific instances or to a specific resource group.

Exceptions

- `INVALID_HANDLE`. The specified handle is not attached to a Data Pump job.
- `INVALID_STATE`. The causes of this exception can be any of the following:
  - No files have been defined for an Export, non-network Import, or SQL_FILE job
  - An `ADD_FILE` procedure has not been called to define the output for a `SQL_FILE` job
  - A `TABLESPACE_DATAFILE` parameter has not been defined for a Transportable Import job
  - A `TABLESPACE_EXPR` metadata filter has not been defined for a Transportable or Tablespace mode Export or Network job
  - The dump file set on an Import or SQL_FILE job was either incomplete or missing a master table specification
- `INVALID_OPERATION`. Unable to restore master table from a dump file set.
- `INTERNAL_ERROR`. An inconsistency was detected when the job was started. Additional information may be available through the `GET_STATUS` procedure.
- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.
- `NO_SUCH_JOB`. The specified job does not exist.

Usage Notes

- When this procedure is called to request that the corresponding job be started or restarted, the state of the job is changed from either the Defining or Idling state to the Executing state.
- If the `SET_PARALLEL` procedure was not called prior to the `START_JOB` procedure, the initial level of parallelism used in the job will be 1. If `SET_PARALLEL` was called prior to the job starting, the `degree` specified by the last `SET_PARALLEL` call determines the parallelism for the job. On restarts, the parallelism is determined by the previous parallel setting for the job, unless it is overridden by another `SET_PARALLEL` call.
- To restart a stopped job, an `ATTACH` function must be performed prior to executing the `START_JOB` procedure.

STOP_JOB Procedure

This procedure terminates a job, but optionally, preserves the state of the job.

Syntax

```
DBMS_DATAPUMP.STOP_JOB (
   handle       IN NUMBER,
   immediate    IN NUMBER DEFAULT 0,
   keep_master IN NUMBER DEFAULT NULL,
   delay       IN NUMBER DEFAULT 60);
```

Parameters

Table 64-27 STOP_JOB Procedure Parameters

Parameter	Description
`handle`	The handle of a job. The current session must have previously attached to the handle through a call to either the `OPEN` or `ATTACH` function. At the end of the procedure, the user is detached from the handle.
`immediate`	If nonzero, the worker processes are aborted immediately. This halts the job quickly, but parts of the job will have to be rerun if the job is ever restarted. If zero, the worker processes are allowed to complete their current work item (either metadata or table data) before they are terminated. The job is placed in a Stop Pending state while the workers finish their current work.
`keep_master`	If nonzero, the master table is retained when the job is stopped. If zero, the master table is dropped when the job is stopped. If the master table is dropped, the job will not be restartable. If the master table is dropped during an export job, the created dump files are deleted.
`delay`	The number of seconds to wait until other attached sessions are forcibly detached. The delay allows other sessions attached to the job to be notified that a stop has been performed. The job keeps running until either all clients have detached or the delay has been satisfied. If no delay is specified, then the default delay is 60 seconds. If a shorter delay is used, clients might not be able to retrieve the final messages for the job through the `GET_STATUS` procedure.

Exceptions

- `INVALID_HANDLE`. The specified handle is not attached to a Data Pump job.
- `INVALID STATE`. The job is already in the process of being stopped or completed.
- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` procedure.
- `NO_SUCH_JOB`. The specified job does not exist.

Usage Notes

- This procedure is used to request that the corresponding job stop executing.
- The termination of a job that is in an Executing state may take several minutes to complete in an orderly fashion.
- For jobs in the Defining, Idling, or Completing states, this procedure is functionally equivalent to the `DETACH` procedure.
- Once a job is stopped, it can be restarted using the `ATTACH` function and `START_JOB` procedures, provided the master table and the dump file set are left intact.
- If the `KEEP_MASTER` parameter is not specified, and the job is in the Defining state or has a mode of Transportable, the master table is dropped. Otherwise, the master table is retained.

WAIT_FOR_JOB Procedure

This procedure runs a job until it either completes normally or stops for some other reason.

Syntax

```
DBMS_DATAPUMP.WAIT_FOR_JOB (
  handle       IN   NUMBER,
  job_state    OUT  VARCHAR2);
```

Parameters

Table 64-28 WAIT_FOR_JOB Procedure Parameters

Parameter	Description
`handle`	The handle of the job. The current session must have previously attached to the handle through a call to either the `OPEN` or `ATTACH` function. At the end of the procedure, the user is detached from the handle.
`job_state`	The state of the job when it has stopped executing; either `STOPPED` or `COMPLETED`.

Exceptions

- `SUCCESS_WITH_INFO`. The procedure succeeded, but further information is available through the `GET_STATUS` API.
- `INVALID_HANDLE`. The job handle is no longer valid.

Usage Notes

This procedure provides the simplest mechanism for waiting for the completion of a Data Pump job. The job should be started before calling `WAIT_FOR_JOB`. When `WAIT_FOR_JOB` returns, the job will no longer be executing. If the job completed normally, the final status will be `COMPLETED`. If the job stopped executing because of a `STOP_JOB` request or an internal error, the final status will be `STOPPED`.