Programming Oracle Data Pump

Sandi Reddick
OraPro.net
Session #446
Email: sandi@orapro.net

About you

- DBAs?
- Developers?
- Experience with exp/imp?
- Experience with expdp/impdp?
- Experience doing imports/exports using OEM?
- PL/SQL?

- Goals
  - Overview of Data Pump API
  - Toolbox addition
- Not goals
  - Learn Data Pump and export/import basics
  - Learn PL/SQL programming

About me

- B.S. in Computer Science
  - University of Central Florida, 1996
- Oracle DBA since 1997, version 7.3.4
  - Lockheed Martin:
    - FBI AFIS, DOD WarSim
  - Hewlett Packard:
    - Instant Delivery, Managed Print Services
  - Fiserv:
    - Branch Suite, TrueImage and Acumen product lines, Credit Union Division
- Previously presented UKOUG
  - Tuning Oracle Text
- Active member of WWOUG

Let’s Get Started!

WHY DBMS_DATAPUMP?

- It is THE Data Pump API
- Used by expdp, impdp and OEM
- Define, execute, monitor and interact with data pump jobs programmatically
- Robust control feature set
  - All of the features available thru command line and HUI interfaces…and MORE!
- More flexible than shell/batch scripts

Prerequisites

- Permissions
  - EXP_FULL_DATABASE role
  - IMP_FULL_DATABASE role
- Directory object
  - at least one required
  - default: DATA_PUMP_DIR
  - read/write granted

DBMS_DATAPUMP Package Overview

Job Control Functions

Function Name	Return type	Description
OPEN	NUMBER	Call to create a new job.
ATTACH	NUMBER	Establish access to a previously created job.

Job Control Procedures

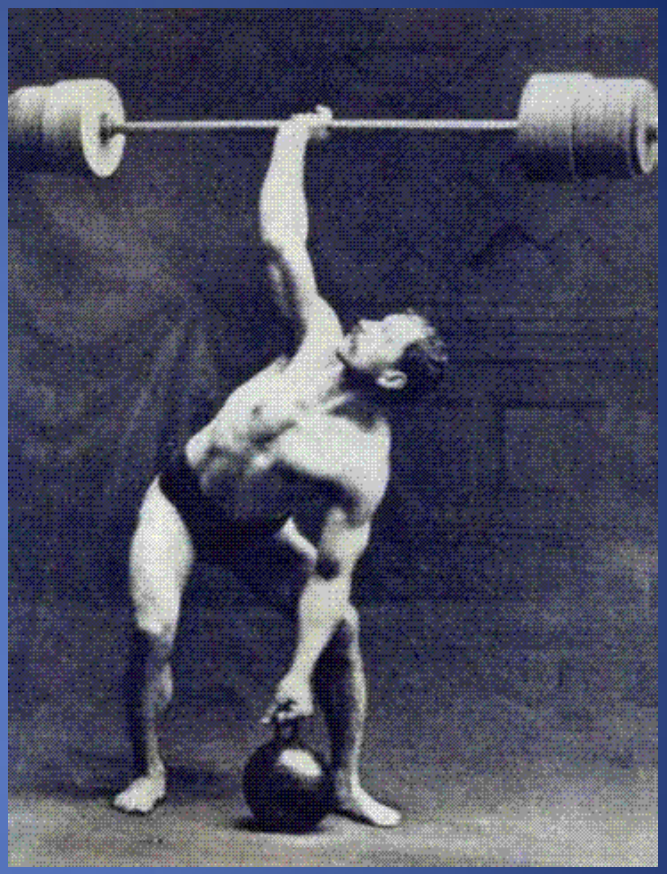
Procedure Name	Description
ADD_FILE	Define dump, log and/or SQL output files.
SET_PARAMETER	Define optional job processing behaviors (ex: encryption, compression, table exists action, …)
SET_PARALLEL	Define maximum number of worker processes than can be spawned for the job. EE only.
START_JOB	Begin or resume execution of a defined job and return program control.
STOP_JOB	Terminate a job, optionally preserving job state info.
WAIT_FOR_JOB	Begin or resume execution of a defined job but do not return program control until job halts.
DETACH	Releases job handle, ending access to job without terminating it.

Filtering and Modification

Procedure Name	Description
DATA_FILTER	Define filters to restrict table row data
METADATA_FILTER	Define filters to restrict objects
METADATA_REMAP	Define object remapping such as schema or tablespaces
METADATA_TRANSFORM	Define object transformations such as storage parameters or OID reassignment.

Getting Information

Procedure Name	Description
GET_DUMPFILE_INFO	Get information about a dump file such as file type, database version, character set.
GET_STATUS	Monitor the progress of a job and retrieve error messages.
LOG_ENTRY	Add text to a log file and optionally display to attached users.



Data Pump API Basics

Required Calls

- Create Data Pump job
- Define job parameters
- Start the job

Optional Calls

- Refine object selection
- Monitor job execution
- Detach from/attach to job
- Stop job
- Restart job

Create the Data Pump job

- Function: OPEN
  - Returns NUMBER data type
  - ‘Handle’ required by future API calls
  - Creates master table

```
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',
        compression     IN NUMBER DEFAULT
                        dbms_datapump.ku$_compress_metadata)
RETURN NUMBER;
```

Create the Data Pump job (cont'd)

- Calling OPEN
  - Required parameters
    - operation => EXPORT | IMPORT | SQL_FILE
    - job_mode => FULL | SCHEMA | TABLE | TABLESPACE | TRANSPORTABLE
  - Usage Example:

```
h1 := dbms_datapump.OPEN(
        operation   => 'EXPORT',
        job_mode    => 'FULL',
        job_name    => 'FULL_DB_EXP');
```

Define the output files

- Procedure: ADD_FILE
  - Define output files
  - Identify input files (import only)
  - Not required for direct imports

```
dbms_datapump.ADD_FILE (
        handle       IN NUMBER,
        filename     IN VARCHAR2,
        directory    IN VARCHAR2 DEFAULT NULL,
        filesize     IN VARCHAR2 DEFAULT NULL,
        filetype     IN NUMBER DEFAULT
                     dbms_datapump.ku$_file_type_dump_file,
        reusefile    IN NUMBER   DEFAULT NULL);
```

Define the output files (cont'd)

- Calling ADD_FILE
  - Required parameters
    - Handle
    - Filename
  - Usage Examples:

```
dbms_datapump.ADD_FILE(
      handle     => h1,
      filename   => 'FULL_DB_EXP.DMP');
dbms_datapump.ADD_FILE(
      handle     => h1,
      filename   => 'FULL_DB_EXP.LOG',
      filetype   => dbms_datapump.ku$_file_type_log_file);
```

Start the job

- Procedure: START_JOB
  - Used to start or restart job
  - Calling changes job state to ‘EXECUTING’

```
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);
```

Start the job (cont'd)

- Calling START_JOB
  - Required parameters
    - Handle
  - Usage Examples:

```
dbms_datapump.START_JOB( handle   => h1);
```

Define Job Parameters

- Procedure: SET_PARAMETER
  - Used to define optional job processing preferences

```
dbms_datapump.SET_PARAMETER (
        handle      IN NUMBER,
        name        IN VARCHAR2,
        value       IN [ VARCHAR2 | NUMBER ] ) ;
```

*Value parameter type overloaded

Define Job Parameters (cont'd)

- SET_PARAMETER Parameter Names

CLIENT_COMMAND
COMPRESSION
DATA_OPTIONS
ENCRYPTION
ENCRYPTION_ALGORITHM
ENCRYPTION_MODE
ENCRYPTION_PASSWORD
ESTIMATE
ESTIMATE_ONLY
FLASHBACK_SCN
FLASHBACK_TIME
INCLUDE_METADATA
PARTITION_OPTIONS
SKIP_UNUSABLE_INDEXES
SOURCE_EDITION
TABLE_EXISTS_ACTION
TABLESPACE_DATAFILE
TARGET_EDITION
TRANSPORTABLE
TTS_FULL_CHECK
USER_METADATA

Define Job Parameters (cont'd)

- Calling SET_PARAMETER
  - Required parameters
    - Handle
    - Name
    - Value
  - Usage Examples:

```
dbms_datapump.SET_PARAMETER(
      handle      => h1,
      name        => 'ESTIMATE_ONLY'
      value       => 1);
dbms_datapump.SET_PARAMETER(
      handle      => h1,
      name        => 'TABLE_EXISTS_ACTION'
      value       => 'TRUNCATE');
```

Demo #1

Full DB Export

Quelle: Deutsche Fotothek

Getting Selective:

Adding Object Filters

Adding Object Filters

- Procedure: METADATA_FILTER
  - Refines job scope
  - Filter types: NAME, SCHEMA, TABLESPACE, INCLUDE PATH, EXCLUDE PATH
  - Filter versions: LIST, EXPR
  - Multiple filters are ANDed together

Adding Object Filters (cont'd)

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

- Calling METADATA_FILTER
  - Required parameters
    - Handle
    - Name (Ex: 'NAME_LIST', 'SCHEMA_EXPR', etc)
    - Value

Adding Object Filters (cont'd)

- Usage Examples:

```
dbms_datapump.METADATA_FILTER(handle  => h1, … … );

name  => 'SCHEMA_EXPR',          value  => 'IN (''SH'')'
name  => 'NAME_EXPR',            value  => 'LIKE ''C%'''
name  => 'SCHEMA_LIST',          value  => '''SH'',''HR'',''BI'''
name  => 'NAME_LIST',            value  => '''CUSTOMERS'',''COST'''
name  => 'TABLESPACE_EXPR',      value  => '=''EXAMPLE'''
name  => 'TABLESPACE_LIST',      value  => '''EXAMPLE'''
name  => 'INCLUDE_PATH_EXPR',    value  => 'LIKE ''/TABLE/%'''
name  => 'EXCLUDE_PATH_EXPR',    value  => '!=''/TABLE/INDEX'''
```

Demo #2

Filtered Schema Export

Getting More Selective: Adding Data Filters

Adding Data Filters

- Procedure: DATA_FILTER
  - Reduces amount of data selected
  - Explicit or sample
  - Used once per table or job

```
dbms_datapump.DATA_FILTER (
        handle                  IN NUMBER,
        name                    IN VARCHAR2,
        value                   IN [ NUMBER | VARCHAR2 | CLOB ] ,
        table_name              IN VARCHAR2 DEFAULT NULL,
        schema_name             IN VARCHAR2 DEFAULT NULL);
```

*Value parameter type overloaded

Adding Data Filters (cont'd)

- Calling DATA_FILTER
  - Required parameters
    - Handle
    - Name => INCLUDE_ROWS | PARTITION_EXPR | PARTITION_LIST | SAMPLE | SUBQUERY
    - Value
  - Usage Examples:

```
dbms_datapump.DATA_FILTER(
      handle      => h1,
      name        => 'SUBQUERY '
      value       => 'WHERE CUST_ID > 50000');
```

Demo #3

Filtered Table Export

Making Changes:
Remap and Transform

Remapping objects

- Procedure: METADATA_REMAP
  - Define remapping of object names, ownership or physical storage
  - Applies to import and SQL File only

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

Remapping objects (cont'd)

- Calling METADATA_REMAP
  - Required parameters
    - Handle
    - Name => REMAP_SCHEMA | REMAP_TABLESPACE | REMAP_DATAFILE | REMAP_TABLE
    - Old value
    - Value
  - Usage Example:

```
dbms_datapump.METADATA_REMAP(
        handle      => h1,
        name        => 'REMAP_SCHEMA',
        old_value   => 'SH',
        value       => 'SCOTT');
```

Demo #4

Remapped Table Import

Transformations

- Procedure: METADATA_TRANSFORM
  - Supports limited type of object definition changes
  - Applies to import and SQL File only

```
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);
```

Transformations (cont'd)

- Calling METADATA_TRANSFORM
  - Required parameters
    - Handle
    - Name => PCTSPACE | SEGMENT_ATTRIBUTES | STORAGE | OID
    - Value
  - Usage Example:

```
dbms_datapump.METADATA_TRANSFORM(
        handle => h1,
        name   => 'STORAGE',
        value  => 1);
```

Demo #5

Transforms and SQL_FILE

Wait! What?

There’s more?

Doing More With DBMS_DATAPUMP

- Encryption
- Compression
- Attach to defining, idling or executing jobs
- Retrieve execution status and progress info
- Adjust the degree of parallelism
- Determine dumpfile version, content, validity
- Define filters conditionally
- Broadcast messages to log files or attached users
- …

Questions?

Thank you!

Email: sandi@orapro.net

Session #446