DUL User's and Configuration Guide V10.2.4.27

Oracle Confidential

DUL and this documentation is Oracle Confidential and for Internal use only.

Table of contents

DUL's PRINCIPLES and FEATURE LIST

STANDALONE C-PROGRAM

DUL is a standalone C program that directly retrieves rows from tables in data files. The Oracle RDBMS software is NOT used at all. DUL does dirty reads, it assumes that every transaction is committed. Nor does it check/require that media recovery has been done.

LAST RESORT

DUL is intended to retrieve data that cannot be retrieved otherwise. It is NOT an alternative for EXP, SQL*Plus etc. It is meant to be a last resort, not for normal production usage.

Before you use DUL you must be aware that the rdbms has many hidden features to force a bad database open. Undocumented init.ora parameters and events can be used to skip roll forward, to disable rollback, disable certain SMON actions, advance the database scn and more.

DATABASE CORRUPT - BLOCKS OK

The database can be corrupted, but an individual data block used must be 100% correct. During all unloading checks are made to make sure that blocks are not corrupted and belong to the correct segment. If during a scan a bad block is encountered, an error message is printed in the loader file and to standard output. Unloading will continue with the next row or block.

ROWS in CLUSTERS/TABLES/INDEXES

DUL can and will only unload index/table/cluster data. It will NOT dump triggers, stored procedures nor create sql scripts for tables or views. (But the data dictionary tables describing them can be unloaded). The data will be unloaded in a format suitable for SQL*Loader or IMP. A matching control file for SQL*Loader is generated as well.

DUL can unload indices and index organized tables. Index unload is usefull to determine how many rows a table should have or to identify the missing rows.

CROSS PLATFORM UNLOADING

Cross-platform unloading is supported. The database can be copied from a different operating system than the DUL-host. (Databases/systems done so far: Sequent/ptx, Vax Vms, Alpha Vms, MVS, HP9000/8xx, IBM AIX, SCO Unix, Alpha OSF/1, Intel Windows NT).

The configuration parameters within "init.dul" will have to be modified to match those of the original platform and O/S rather than the platform from which the unload is being done.

ROBUST

DUL will not dump, spin or hang no matter how badly corrupted the database is.

(NEARLY) ALL ORACLE FEATURES SUPPORTED

Full support for all database constructs: row chaining, row migration, hash/index clusters, longs, raws, rowids, dates, numbers, multiple free list groups, segment high water mark, NULLS, trailing NULL columns, and unlimited extents, new block layout of Oracle8, partitioned tables.

Later additions are lobs, compressed indexes, 9ir2 compressed tables. Varrays and ADTs (user defined objects) are partly supported in sql*loader mode.

ASM is fully supported, files can be extracted from an asm disk group. No mounted ASM instance is used, the disks are accessed directly. Non default asm allocation unit sizes are supported.

Data can be recovered from export dump files with the unexp command suite. Some initial work has been done for unpump to support data pump files.

SUPPORTED RDBMS VERSIONS

DUL should work with all versions starting oracle 6. DUL has been tested with versions from 6.0.26 up to 10.2. Even the old block header layout (pre 6.0.27.2) is supported.

MULTI BYTE SUPPORT

DUL itself is essentially a single byte application. The command parser does not understand multi byte characters, but it is possible to unload any multi byte database. For all possible caveats there is a work around.

DUL can optionally convert to UTF8. This is for NCLOBS that are stored in UTF16.

RESTRICTIONS

MLS LABELS

Multi Level Security Lables of trusted oracle are not supported.

(LONG) RAW

DUL can unload (long) raws. Nowadays there is suitable format in SQL*Loader to preserve all long raws. So Long raws and blobs can be unloaded in both modes.

ORACLE8 OBJECT OPTION AND LOBS

Nested tables are not yet supported, if they are needed let me know and it will be added. Varrays and ADTs are supported, also those that are stored as a kernel lob. CLOBS, NCLOBS are supported both in SQL*Loader mode and in exp mode. BLOBS are best handled in exp mode, the generated hex format in SQL*Loader mode is not loaded correctly currently.

PORTABLE

DUL can be ported to any operating system with an ANSI-C compiler. DUL has been ported to many UNIX variants, VMS and WindowsNT. Currently all builds are done using gcc and a cross compiler environment on Linux

RDBMS INTERNALS

A good knowledge of the Oracle RDBMS internals is a pre requisite to be able to use DUL successfully. For instance the Data Server Internals (DSI) courses give a good foundation. There is even a module dedicated to DUL

SETTING UP and USING DUL

CONFIGURATION FILES

There are two configuration files for DUL. "init.dul" contains all configuration parameters. (size of caches, details of header layout, oracle block size, output file format) In the control file, "control.dul", the database data file names and the asm disks can be specified.

DATA DICTIONARY AVAILABLE

The Oracle data dictionary is available if the data files which made up the SYSTEM TableSpace are available and useable. The number which Oracle assigned to these files and the name you have given them, which does not have to be the original name which Oracle knew, must be included in the "control.dul" file. You also need to eventually include the file numbers and names of any files from other TableSpaces for which you wish to eventually unload TABLES and their data. The lack of inclusion of these files will not affect the data dictionary unload step but it will affect later TABLE unloading.

USING DUL WHEN *USER$, OBJ$, TAB$ and COL$* CAN BE UNLOADED

Steps to follow:

1. configure DUL for the target database. This means creating a correct init.dul and control.dul. The SYSTEM TableSpace's data file numbers and names must be included within the control.dul file along with any data files for TableSpaces from which you wish to unload TABLEs and their data. For Oracle8 and higher the tablespace number and the relative file number must be specified for each datafile.
2. Use the " BOOTSTRAP; " command to prepare for unloading. The bootstrap process will find a compatibility segment, find the bootstrap$ table unload The old " dul dictv7.ddl"re no longer needed.
3. Unload the tables for which data files have been included within the "control.dul" file. Use one of the following commands:
   - "UNLOAD TABLE *[owner>.]table* ; (do not forget the semicolon)
     - This will unload the one table definition and the table's data.
   - "UNLOAD USER *user name* ;
     - This unloads all tables and data for the specified user.
   - "UNLOAD DATABASE ;
     - This unloads all of the database tables available. (except the user SYS).

NO DATA DICTIONARY AVAILABLE

If data files are not available for the SYSTEM TableSpace the unload can still continue but USER, TABLE and COLUM names will not be known. Identifying the tables can be an overwhelming task. But it can be (and has been) done. You need in depth knowledge about your application and the application tables. Column types can be guessed by DUL, but table and column names are lost. Any old SYSTEM tablespace from the same database but weeks old can be of great help!. Most of the information that DUL uses does not change. (only the dataobj# is during truncate or index rebuild)

USING DUL WITHOUT SYSTEM TABLESPACE

Steps to follow:

1. configure DUL for the target database. This means creating a correct init.dul and control.dul. (See Port specific parameters). In this case control.dul file will need the numbers and names of datafiles from which TABLEs and data will be unloaded but it does not require the SYSTEM TableSpace's information.
2. SCAN DATABASE; : scan the database, build extent and segment map
3. SCAN TABLES; or SCAN EXTENTS; : gather row statistics
4. Identify the lost tables from the output of step 3.
5. UNLOAD the identified tables.

AUTOMATED SEARCH

To ease the hunt for the lost tables: the scanned statistical information in seen_tab.dat and seen_col.dat can be loaded into a fresh database. If you recreate the tables (Hopefully the create table scripts are still available) then structure information of a "lost" table can be matched to the "seen" tables scanned information with two SQL*Plus scripts. (fill.sql and getlost.sql).

HINTS AND PITFALLS

- Names are not really relevant for DUL, only for the person who must load the data. But the unloaded data does not have any value, if you do not know from which table it came.
- The guessed column types can be wrong. Even though the algorithm is conservative and decides UNKNOWN if not sure.
- Trailing NULL columns are not stored in the database. So if the last columns only contain NULL's than the scanner will NOT find them. (During unload trailing NULL columns are handled correctly).
- When a table is dropped, the description is removed from the data dictionary only. The data blocks are not overwritten unless they are reused for a new segment. So the scanner software can see a table that has been dropped.
- Tables without rows will go unnoticed.
- Newer objects have a higher object id than older objects. If an table is recreated, or if there is a test and a production version of the same table the object id can be used to decide.

DDL (DUL Description Language) UNLOAD STATEMENT OVERVIEW

DUL uses an SQL like command interface. There are DDL statements to unload extents, tables, users or the entire database. Data dictionary information required can be specified in the ddl statements or taken from the previously unloaded data dictionary. The following three statements will unload the DEPT table. The most common form is if the data dictionary and the extent map are available:

```
        UNLOAD TABLE scott.dept;
```

All relevant information can be specified in the statement as well:

```
        REM Columns with type in the correct order
        REM The segment header loaction in the storage clause
        UNLOAD TABLE dept( deptno NUMBER, dname CHAR, loc CHAR)
               STORAGE( EXTENTS ( FILE 1 BLOCK 1205 ));
```

Oracle version 6:

```
        REM version 6 data blocks have segment header location in each block
        ALTER SESSION SET USE_SCANNED_EXTENT_MAP = TRUE;
        UNLOAD TABLE dept( deptno NUMBER, dname CHAR, loc CHAR)
               STORAGE( EXTENTS ( FILE 1 BLOCK 1205 ));
```

Oracle7:

```
        REM Oracle7 data blocks have object id in each block

        ALTER SESSION SET USE_SCANNED_EXTENT_MAP = TRUE;
        UNLOAD TABLE dept( deptno NUMBER, dname CHAR, loc CHAR)
               STORAGE( OBJNO 1501 );
```

DUL'S OUTPUT FORMAT.

Only complete good rows are written to the output file. For this each row is buffered. The size of the buffer can changed with the init.dul parameter BUFFER. There is no speed gained with a high BUFFER parameter, it should just be big enough to hold a complete row. Incomplete or bad rows are not written out. The FILE_SIZE_IN_MB init.dul parameter can be used to split the output (at a proper boundary) into multiple files, each file can be loaded individually.

There are three different modes of output format.

- Export mode
- SQL*Loader mode: stream data files
- SQL*Loader mode: Fixed physical record data files

EXPORT MODE

The generated file is completely different from a table mode export generated by EXP! The file is the minimal format that IMP can load. For each table a separate IMP loadable file will be generated. It is a single table dump file. It contains a header an insert table statement and the table data. Table grants, storage clauses, or triggers will not be included. An minimal create table statement is included (no storage clause just column names and types without precision). The character set indication in the file in the generated header is V6 style. It is set to mean ASCII based characterset.

To enable export mode, set the init.dul parameter EXPORT_MODE to TRUE.

As the generated pseudo dump file does not contain character set information set NLS_LANG to match that of the original database. In export mode no character set conversion is done.

SQL*LOADER MODES

The data in the is either not converted at all, or everthing is converted to UTF8 if LDR_OUTPUT_IN_UTF8 is set. This setting is required in mixed character set environments as the contents of a data file must have a single character set.<\P>

When loading the data you probably need to set NLS_LANG to match that of the original database to prevent unwanted character set conversion.

For both SQL*Loader output formats the columns will be space separated and enclosed in double quotes. Any double quote in the data will be doubled. SQL*Loader recognizes this and will load only one. The character used to enclose the columns can be changed from double quote to any character you like with the init.dul parameter LDR_ENCLOSE_CHAR.

There are two styles of physical record organization:

Stream Mode

Nothing special is done in stream mode, a newline is printed after each record. This is a compact format and can be used if the data does not contain newline characters. To enable stream mode set LDR_PHYS_REC_SIZE = 0 in init.dul.

Fixed Physical Records

This mode is essential if the data can contain newlines. One logical record, one comlete row, can be composed of multiple physical records. The default is record length is 81, this fits nicely on the screen of a VT220. The physical record size can be specified with LDR_PHYS_REC_SIZE in init.dul.

OUTPUT FILE NAMES

The file names generated are: *owner name_table name*.ext. The extension is ".dmp" for IMP loadable files. ".dat" and ".ctl" are used for the

SQL*Loader datafile and the control file. To prevent variable substitution and other unwanted side effects, strange characters are stripped.(Only alpha numeric and '_' are allowed).

If the FILE parameter is set the generated names will be FILEnnn.ext. This possibility is a work around if the file system does not support long enough file names. (Old windows with 6.3 filename format)

SOME DUL INTERNALS

REQUIRED INFORMATION

To unload table data from a database block the following information must be known:

1. Column/Cluster Information: The number and type of the columns. For char or varchar columns the maximum length as well. The number of cluster columns and the table number in the cluster. This information can be supplied in the unload statement or it can be taken from the previously unloaded USER$, OBJ$, TAB$ and COL$.
2. Segment/Extent information: When unloading a table the extent table in the data segment header block is used to locate all data blocks. The location of this segment header block (file number and block number) is taken from the data dictionary or can be specified in the unload statement. If the segment header is not correct/available then another method must be used. DUL can build its own extent map by scanning the whole database. (in a separate run of DUL with the scan database statement.)

BINARY HEADERS

C-Structs in block headers are not copied directly, they are retrieved with specialized functions. All offsets of structure members are programmed into DUL. This approach makes it possible to cross-unload. (Unload an MVS created data file on an HP) Apart from byte order only four layout types have been found so far.

1. Vax VMS and Netware : No alignment padding between structure members.
2. Korean Ticom Unix machines : 16 bit alignment of structure members.
3. MS/DOS 16 bit alignment and 16 bit wordsize.
4. Rest of the world (Including Alpha VMS) structure member alignment on member size.

MACHINE DEPENDENCIES

Machine dependencies (of the database) are configurable with parameters:

- Order of bytes in a word (big/little endian).
- Number of bits for the low part of the FILE# in a DBA (Block Address).
- Alignment of members in a C-struct.
- Number of blocks or bytes before the oracle file header block.
- Size of a word used in the segment header structure.

UNLOADING THE DATA DICTIONARY

DUL can use the data dictionary of the database to be unloaded if the files for it exist and are uncorrupted. For the data dictionary to be used, internal tables must be unloaded first to external files: (USER$, OBJ$, TAB$ and COL$). The bootstrap command will find and unload the required tables.

DDL (DUL DESCRIPTION LANGUAGE) SPECIFICATION

```
[ ALTER SESSION ] SET init.dul parameter =  value ;
     Most parameters can be changed on the fly.

BOOTSTRAP [LOCATE | GENERATE | COMPLETE
          | UNLOAD   Bootstrap$ segment header block address ];
     Bootstraps the data dictionary. Default is COMPLETE.
     LOCATE finds and unloads the bootstrap$ table.
     GENERATE builds a ddl file based on inforation in the cache.
     COMPLETE is in fact LOCATE, followed by GENERATE (two times)

COMMIT;
     Writes the changed block to the data file.

CREATE BLOCK INDEX  index_name  ON  device ;
```

A block index contains address of valid oracle blocks found in a corrupt file system. Useful to merge multiple disk images or to unload from corrupted file systems. This is only useful in extreme file system corruption scenarios.

```
DESCRIBE  owner_name  . table_name ;

DUMP [ TABLESPACE  tablespace_no ]
     [ FILE  file_no  ]
     [ BLOCK  block_no  ]
     [ LEVEL  level_no  ] ;
     Not a complete blockdump, mainly used for debugging.
     The block address is remembered.

EXTRACT  asm file name  to  output file name  ;
     Copies any ASM file from a disk group to the file system.
     (there was a problem with online redologs this needs more testing)

MERGE block_index INTO [  segment  ];
```

The merge command uses the information in the index file to locate possible data blocks it looks for a combination of file numbers and object id, each candidate block is compared to the current block in the datafile. If the current block is bad, or has an older scn the candidate will will be written into the datafile. This is only useful in extreme file system corruption scenarios.

```
REM any_text_you_like_till_End_Of_Line : comment
REM  NOT allowed inside ddl statements. ( To avoid a two layer lexical scan).

ROLLBACK; # Cancels the UPDATE statements.

SHOW     DBA  dba ;                # dba -> file_no block_no calculator
           | DBA  rfile_no block_no ;  # file_no block_no -> dba calculator
           | SIZES ;                   # show some size of important structs
           | PARAMETER;                # shows the values of all parameters
           | LOBINFO;                  # lob indexes found with SCAN DATABASE
       | DATAFILES;                # summary of configured datafiles
       | ASM DISKS;                # summary of configured asm disks
       | ASM FILES;                # summary of configured datafiles on asm
       | ASM FILE  cid      # extent information for asm file

UNEXP [TABLE] [  owner  . ]  table name
       (  column list  ) [ DIRECT ]
       DUMP FILE  dump file name
       FROM  begin offset  [ UNTIL  end offset  ]
       [ MINIMUM  minimal number of columns  COLUMNS ] ;

       To unload data from a corrupted exp dump file. No special setup
       or configuration is required, just the compatible parameter.
       The start offset should be where a row actually begins.

UNPUMP
       To unload data from a corrupted expdp (datapump) dump file.
       This is still work in progress, the basic commands work
       but rather complex to use. Contact me if this is needed.

UNLOAD DATABASE;

UNLOAD USER user_name;

UNLOAD [TABLE]  [  schema_name . ]  table_name
              [ PARTITION(  partition_name ) ]
              [ SUBPARTITION(  sub_partition_name ) ]
              [ (  column_definitions ) ]
              [  cluster_clause  ]
              [  storage_clause  ] ;

UNLOAD EXTENT  table_name
              [ (  column_definitions  ) ]
              [ TABLESPACE  tablespace_no  ]
              FILE  extent_start_file_number
              BLOCK extent_start_block_number
              BLOCKS  extent_size_in oracle_blocks ;

UNLOAD LOB SEGMENT FOR [  schema_name . ]  table_name   [ (  column name  ) ] ;

UNLOAD LOB SEGMENT STORAGE ( SEGOBJNO data obj#) ;

UPDATE [ block_address ] SET UB1|UB2|UB4 @ offset_in_block = new_value ;
UPDATE [ block_address ] SET  block element name  = new_value ;
        Now and then we can repair something.
        Patches the current block and dumps it.
        You can issue multiple UPDATE commands.
        Block is not written yet, use COMMIT to write.

storage_clause ::=
        STORAGE ( storage_specification  [ more_storage_specs ] )

storage_specification ::=
        OBJNO object_id_number
|       TABNO cluster_table_number
|       SEGOBJNO cluster/data_object_number       /* v7/v8 style data block id */
|       FILE  data_segment_header_file_number     /* v6 style data block id */
        BLOCK data_segment_header_block_number )
|       any_normal_storage_specification_but_silently_ignored
```

```
SCAN DATABASE;
```

Scans all blocks of all data files. Two or three files are generated:

1. SEG.dat information of found segment headers (index/cluster/table): (object id, file number, and block number).
2. EXT.dat information of contiguous table/cluster data blocks. (object id(V7), file and block number of segment header (V6), file number and block number of first block, number of blocks, number of tables)
3. SCANNEDLOBPAGE.dat information for each lob datablock, this file (optional, only if init.dul:SCAN_DATABASE_SCANS_LOB_SEGMENTS=TRUE) can possibly be huge. Also the required memory size can be problematic. The purpose is twofold: 1: to possibly work around corrupt lob indexes during unload table. 2: unload lob segments (for deleted lobs or lob segments without lob index or parent table) Meaning of the fields in SCANNEDLOBPAGE.dat: (segobj#, lobid, fat_page_no, version(wrap, base), ts#, file#, block#)

```
SCAN DUMP FILE  dump file name
        [ FROM  begin offset  ]
        [ UNTIL  end offset  ];

        Scans an  export dump file to produce to provide the
        create/insert statements and the offsets in the dump file.

SCAN LOB SEGMENT    storage clause ;
SCAN LOB SEGMENT FOR  table name  [.  column name] ;
        Scans the lob segment to produce LOBPAGE.dat information,
        but then for this segment only. Probably quicker and
        smaller. For partitioned objects use scan database.

SCAN TABLES;
        Uses SEG.dat and EXT.dat as input.
        Scans all tables in all data segments (a header block and at least one
        matching extent with at least 1 table).

SCAN EXTENTS;
        Uses SEG.dat and EXT.dat as input.
        All extents for which no corresponding segment header has been found.
        (Only useful if a tablespace is not complete, or a segment header
        is corrupt).

EXIT QUIT and EOF all cause DUL to terminate.
```

DDL (DUL DESCRIPTION LANGUAGE) DESCRIPTION

Rules for UNLOAD EXTENT and UNLOAD TABLE:

Extent Map

UNLOAD TABLE requires an extent map. In 99.99% of the cases the extent map in the segment header is available. In the rare 0.01% that the segment header is lost an extent map can be build with the scan database command. The self build extent map will ONLY be used during an unload if the parameter USE_SCANNED_EXTENT_MAP is set to TRUE.

All data blocks have some ID of the segment they belong to. But there is a fundamental difference between V6 and V7. Data blocks created by Oracle version 6 have the address of the segment header block. Data blocks created by Oracle7 have the segment object id in the header.

Column Specification

The column definitions must be specified in the order the columns are stored in the segment, that is ordered by col$.segcol#. This is not necessarily the same order as the columns where specified in the create table statement. Cluster columns are moved to the front, longs to the end. Columns added to the table with alter table command, are always stored last.

Unloading a single extent

UNLOAD EXTENT can be used to unload 1 or more adjacent blocks. The extent to be unloaded must be specified with the STORAGE clause: To specify a single extent use: STORAGE (EXTENTS(FILE *fno* BLOCK *bno* BLOCKS *#blocks*)) (FILE and BLOCK specify the first block, BLOCKS the size of the extent)

DUL specific column types

There are two extra DUL specific data types:

1. IGNORE: the column will be skipped as if it was not there at all.
2. UNKNOWN: a heuristic guess will be made for each column.

In SQL*Loader mode there are even more DUL specific data types:

1. HEXRAW: column is HEX dumped.
2. LOBINFO: show some information from LOB locators .
3. BINARY NUMBER: Machine word as used in a LOB index.

Identifying USER$, OBJ$, TAB$ and COL$

DUL uses the same bootstrap procedure as the rdbms. That is it uses the root dba from the system datafile header to locate the bootstrap$ table. Depending on the version this root dba is either the location of the compatibility segment containing the bootstrap$ address or for the newer versions the address of the bootstrap$ table itself. The bootstrap$ table is unloaded and its contents is parsed to find the first four tables (USER$, OBJ$, TAB$ and COL$). The other tables are unloaded based on information in these first four.

DESCRIPTION OF SCAN COMMANDS

SCAN TABLES and SCAN EXTENTS scan for the same information and produce similar output. ALL columns of ALL rows are inspected. For each column the following statistics are gathered:

- How often the column is seen in a data block.
- The maximum internal column length.
- How often the column IS NULL.
- How often the column consists of at least 75% printable ascii.
- How often the column consists of 100% printable ascii.
- How often the column is a valid oracle number.
- How often the column is a nice number. (not many leading or trailing zero's)
- How often the column is a valid date.
- How often the column is a possible valid rowid.

These statistics are combined and a column type is suggested. Using this suggestion five rows are unloaded to show the result. These statistics are dumped to two files (seen_tab.dat and seen_col.dat). There are SQL*Loader and SQL*Plus scripts available to automate a part of the identification process. (Currently known as the getlost option).

DESCRIBE

There is a describe command. It will show the dictionary information for the table, available in DUL's dictionary cache.

DUL STARTUP SEQUENCE

During startup DUL goes through the following steps:

- the parameter file "init.dul" is processed.
- the DUL control file (default "control.dul") is scanned.
- Try to load dumps of the USER$, OBJ$, TAB$ and COL$ if available into DUL's data dictionary cache.
- Try to load seg.dat and col.dat.
- Accept DDL-statements or run the DDL script specified as first arg.

DUL parameters to be specified in init.dul:

ALLOW_TRAILER_MISMATCH
BOOLEAN
Strongly discouraged to use, will seldom produce more rows. Use only if you fully understand what it means and why you want it. skips the check for correct block trailer. The blocks failing this test are split of corrupt. But it saves you the trouble to patch some blocks.

ALLOW_DBA_MISMATCH
BOOLEAN
Strongly discouraged to use, will seldom produce more rows. Use only if you fully understand what it means and why you want it. Skips the check for correct block address. The blocks failing this test are probably corrupt. But it saves you the trouble to patch some blocks.

ALLOW_OTHER_OBJNO
BOOLEAN
If your dictionary is older than your datafiles then the data object id's can differ for truncated tables. With this parameter set to true it will issue a warning but use the value from segment header. All other blocks are fully checked. This is for special cases only.

ASCII2EBCDIC
BOOLEAN
Must (var)char fields be translated from EBCDIC to ASCII. (For unloading MVS database on a ASCII host)

BUFFER
NUMBER (bytes)
row output buffer size used in both export and SQL*Loader mode. In each row is first stored in this buffer. Only complete rows without errors are written to the output file.

COMPATIBLE
NUMBER
Database version , valid values are 6, 7, 8 or 9. This parameter must be specified

CONTROL_FILE

TEXT

Name of the DUL control file (default: "control.dul").

DB_BLOCK_SIZE

NUMBER

Oracle block size in bytes (Maximum 32 K)

DC_COLUMNS

NUMBER

DC_OBJECTS

NUMBER

DC_TABLES

NUMBER

DC_USERS

NUMBER

Sizes of dul dictionary caches. If one of these is too low the cache will be automatically resized.

EXPORT_MODE

BOOLEAN

EXPort like output mode or SQL*Loader format

FILE

TEXT

Base for (dump or data) file name generation. Use this on 8.3 DOS like file systems

FILE_SIZE_IN_MB

NUMBER (Megabytes)

Maximum dump file size. Dump files are split into multiple parts. Each file has a complete header and can be loaded individually.

LDR_ENCLOSE_CHAR

TEXT

The character to enclose fields in SQL*Loader mode.

LDR_PHYS_REC_SIZE

NUMBER

Physical record size for the generated loader datafile.

LDR_PHYS_REC_SIZE = 0 No fixed records, each record is terminated with a newline.

LDR_PHYS_REC_SIZE > 2: Fixed record size.

MAX_OPEN_FILES

Maximum # of database files that are concurrently kept open at the OS level.

OSD_BIG_ENDIAN_FLAG

Byte order in machine word. Big Endian is also known as MSB first. DUL sets the default according to the machine it is running on. For an explanation why this is called Big Endian, you should read Gullivers Travels.

OSD_DBA_FILE_BITS

File Number Size in DBA in bits. Or to be more precise the size of the low order part of the file number.

OSD_FILE_LEADER_SIZE

bytes/blocks added before the real oracle file header block

OSD_C_STRUCT_ALIGNMENT

C Structure member alignment (0,16 or 32). The default of 32 is correct for most ports.

OSD_WORD_SIZE

Size of a machine word always 32, except for MS/DOS(16)

PARSE_HEX_ESCAPES

Boolean default FALSE

Use \\xhh hex escape sequences in strings while parsing. If set to true then strange characters can be specified using escape sequences. This feature is also for specifying multi-byte characters.

USE_SCANNED_EXTENT_MAP

BOOLEAN

Use the scanned extent map in ext.dat when unloading a table. The normal algorithme uses the extent map in the segment header. This parameter is only useful if some segment headers are missing or incorrect.

WARN_RECREATE_FILES

BOOLEAN (TRUE)

Set to FALSE to suppress the warning message if an existing file is overwritten.

WRITABLE_DATAFILES

BOOLEAN (FALSE)

Normal use of DUL will only read the database files. However the UPDATE and the SCAN RAW DEVICE will write as well. The parameter is there to prevent accidental damage.

SAMPLE init.dul :

```
# sample init.dul configuration parameters
# these must be big enough for the database in question
# the cache must hold all entries from the dollar tables.
dc_columns = 200000
dc_tables = 10000
dc_objects = 10000
```

```
dc_users = 40

# OS specific parameters
osd_big_endian_flag = false
osd_dba_file_bits = 10
osd_c_struct_alignment = 32
osd_file_leader_size = 1

# database parameters
db_block_size = 8k

# loader format definitions
LDR_ENCLOSE_CHAR = "
LDR_PHYS_REC_SIZE = 81
```

Configuring the port dependent parameters

Collection of known Parameters

There is a list of osd (Operating System Dependend) parameters for almost every platform. If your platform is not in the list you can use the suggestions below to determine the parameters. (And then please inform me so I can add them to the list.)

osd_big_endian_flag

big endian or little endian (byte order in machine words): HP, SUN and mainframes are generally big endian: OSD_BIG_ENDIAN_FLAG = TRUE. DEC and Intel platforms are little endian: OSD_BIG_ENDIAN_FLAG = FALSE. The default is correct for the platform where DUL is running on.

There is no standard trick for this, the following might work on a unix system:

```
  echo dul | od -x
  If the output is like:
     0000000 6475 6c0a
     0000004
  You are on a big endian machine (OSD_BIG_ENDIAN_FLAG=TRUE).

  If you see:
     0000000 7564 0a6c
     0000004
  This is a little endian machine (OSD_BIG_ENDIAN_FLAG=FALSE).
```

osd_dba_file_bits

The number of bits in a dba used for the low order part of file number. Perform the following query:

```
  SQL> select dump(chartorowid('0.0.1')) from dual;

  Typ=69 Len=6: 8,0,0,0,0,0    ->       osd_dba_file_bits =  5 (SCO)
  Typ=69 Len=6: 4,0,0,0,0,0    ->       osd_dba_file_bits =  6 (Sequent , HP)
  Typ=69 Len=6: 1,0,0,0,0,0    ->       osd_dba_file_bits =  8 (NCR,AIX)
  Typ=69 Len=6: 0,16,0,0,0,0   ->       osd_dba_file_bits = 12 (MVS)
  Typ=69 Len=10: 0,0,0,0,0,64,0,0,0,0      osd_dba_file_bits = 10 (Oracle8)
```

OSD_C_STRUCT_ALIGNMENT

Structure layout in data file headers. 0: No padding between members in a C-struct (VAX/VMS only) 16: Some korean ticom machines and MS/DOS 32: Structure members are member size aligned. (All others including ALPHA/VMS) Check the following query:

```
SELECT * FROM v$type_size
WHERE type IN ( 'KCBH', 'KTNO', 'KCBH', 'KTBBH', 'KTBIT', 'KDBH'
              , 'KTECT', 'KTETB', 'KTSHC') ;
```

In general osd_c_struct_alignment = 32 and the following output is expected:

```
K        KTNO     TABLE NUMBER IN CLUSTER                   1
KCB      KCBH     BLOCK COMMON HEADER                      20
KTB      KTBIT    TRANSACTION VARIABLE HEADER              24
KTB      KTBBH    TRANSACTION FIXED HEADER                 48
KDB      KDBH     DATA HEADER                              14
KTE      KTECT    EXTENT CONTROL                           44
KTE      KTETB    EXTENT TABLE                              8
KTS      KTSHC    SEGMENT HEADER                            8

8 rows selected.
```

For VAX/VMS and Netware ONLY osd_c_struct_alignment = 0 and this output is expected:

```
COMPONEN TYPE     DESCRIPTION                      SIZE
-------- -------- -------------------------------- ----------
```

```
K        KTNO     TABLE NUMBER IN CLUSTER                  1
KCB      KCBH     BLOCK COMMON HEADER                     20
KTB      KTBIT    TRANSACTION VARIABLE HEADER             23
KTB      KTBBH    TRANSACTION FIXED HEADER                42
KDB      KDBH     DATA HEADER                             14
KTE      KTECT    EXTENT CONTROL                          39
KTE      KTETB    EXTENT TABLE                             8
KTS      KTSHC    SEGMENT HEADER                           7

8 rows selected.
```

If there is a different list this will require some major hacking and sniffing and possibly a major change to DUL. (Email Bernard.van.Duijnen@oracle.com)

osd_file_leader_size

Number of blocks/bytes before the oracle file header. Unix datafiles have an extra leading block (file size, block size magic number) A large number (> 100) is seen as a byte offset, a small number is seen as a number of oracle blocks.

```
Unix    :       osd_file_leader_size = 1
Vms     :       osd_file_leader_size = 0
Desktop :       osd_file_leader_size = 1 (or 512 for old personal oracle)
Others  :       Unknown ( Use Andre Bakker's famous PATCH utility to find out)
                An Oracle7 file header block starts with the pattern 0X0B010000.
```

You can add an additional byte offset in control.dul in the optional third field (for instance for AIX or DEC UNIX data files on raw device)

Control file syntax specification

A control file (default name "control.dul") is used to specify asm disks, block indexes and the data file names. The format of the control has been extended

Currently there are three types of specifications in the DUL control file. Each entry on a separate line. The asm disks must precede the asm files.

```
  control_file_line ::= asm_disk_spec | file_piece_spec | block_index_spec
```

If COMPATIBLE is 10 or higher you can also specify asm disks. Its generally sufficent to specify the device name. All properties are automatically retrieved by header inspection. The full syntax is only needed when header inspection is impossible, that is for disks with corrupt headers. The syntax is:

```
DISK  device name [  disk group options  ]

 disk group option  ::= GROUP  disk group name
                          | DISK_NO  disk number in group
                          | F1B1  File1 Block1 location
```

A block index is a way to access oracle blocks on corrupt file systems. In general a corrupt file system is not wiped out, its not empty. Due to the specific layout of oracle blocks it is possible to datablocks an store their location in the block index. See also the *create block index command* . A *block_index_name* is a normal identifier, it is used to construct an unique file name.

```
BLOCK INDEX  block_index_name


 Each entry can contain a part of a datafile.
The smallest unit is a single data block.
This way it is possible to split datafiles that are too big for DUL
in parts where each part is smaller than 2GB.




 In general it is sufficient to specify the file name.
Even for a single block.
If compatible is 10 or higher the file numbers and the tablespace
numbers will be read from the file header.




If the specified details are different from the file header
DUL will give a warning but use your specification.
This is  to be able to unload files with a corrupted header block.
For debugging it is possible to dump the file header.




 The optional extra leader offset is an extra byte offset,
that will be added to all lseek() operations for that datafile.
This makes it possible to skip over the extra 4k block for some
AIX raw devices, or the extra 64K on Tru64 on raw devices
```

```
  file_piece_spec ::=
         [ [ tablespace_no ] relative_file_number]data_file_name
         [ optional extra leader offset ]
         [ startblock block_no ]
         [ endblock block_no ]
```

Examples

```
   # AIX version 7 example with one file on raw device
   1 /usr/oracle/dbs/system.dbf
   8 /dev/rdsk/data.dbf 4096
```

```
   # Oracle8 example with a datafile split in multiple parts, each part smaller than 2GB
   0  1 /fs1/oradata/PMS/system.dbf
   1  2 /tmp/huge_file_part1 startblock 1 endblock 1000000
   1  2 /tmp/huge_file_part2 startblock 1000001 endblock 2000000
   1  2 /mnt3/huge_file_part3 startblock 2000001 endblock 2550000
```

```
# ASM disks for two disk groups
disk /media/maxtor/asm/dgn1
disk /media/maxtor/asm/dgn2
disk /media/maxtor/asm/dgn3
disk /media/maxtor/asm/dgn4
disk /media/maxtor/asm/dgodd

# system datafile in the first asm disk group
+DGN/db102/datafile/system.257.621616979

# users datafile in a different disk group
+DGODD/db102/datafile/users.257.621616683

# a so called big file tablespace, use 1024 for the file#
 8 1024 /home/oracle/v102/dbs/bigfilets

# Or let DUL find out itself from the header
 /home/oracle/v102/dbs/bigfilets

# one tablespace with a different block size
/home/oracle/v102/dbs/ts16k.dbf block_size 16k

# or let DUL find out by header inspection
/home/oracle/v102/dbs/ts16k.dbf
```

Sample unload session: data dictionary usable for DUL

1. create a suitable "init.dul"

2. create a control.dul

```
        sqlplus /nolog
        connect / as sysdba
        startup mount
        set trimspool on pagesize 0 linesize 256 feedback off
        column name format a200
        spool control.dul
        select ts#, rfile#, name from v$datafile;
        exit
```

   edit the result

```
    For Oracle8 a different query must be used:
       select ts#, rfile#, name from v$datafile;
```

3. start DUL and bootstrap;

```
    $ dul

    Data UnLoader 10.2.1.16 - Oracle Internal Only - on Thu Jun 28 11:37:24 2007
    with 64-bit io functions

    Copyright (c) 1994 2007 Bernard van Duijnen All rights reserved.

     Strictly Oracle Internal use Only
    DUL> bootstrap;
    Probing file = 1, block = 377
    . unloading table                BOOTSTRAP$      57 rows unloaded
    DUL: Warning: Dictionary cache DC_BOOTSTRAP is empty
    Reading BOOTSTRAP.dat 57 entries loaded
    Parsing Bootstrap$ contents
    DUL: Warning: Recreating file "dict.ddl"
    Generating dict.ddl for version 10
     OBJ$: segobjno 18, file 1
     TAB$: segobjno 2, tabno 1, file 1
     COL$: segobjno 2, tabno 5, file 1
     USER$: segobjno 10, tabno 1, file 1
    Running generated file "@dict.ddl" to unload the dictionary tables
    . unloading table                      OBJ$   52275 rows unloaded
    . unloading table                      TAB$    1943 rows unloaded
    . unloading table                      COL$   59310 rows unloaded
    . unloading table                     USER$      70 rows unloaded
    Reading USER.dat 70 entries loaded
    Reading OBJ.dat
     52275 entries loaded and sorted 52275 entries
    Reading TAB.dat 1943 entries loaded
    Reading COL.dat 59310 entries loaded and sorted 59310 entries
    Reading BOOTSTRAP.dat 57 entries loaded
    ...
    Some more messages for all the other TABLES
    ...
    Database character set is WE8ISO8859P1
    Database national character set is AL16UTF16
    DUL> unload user SCOTT;
    About to unload SCOTT's tables ...
    . unloading table                       EMP      14 rows unloaded
```

Example unload session: data dictionary UNUSABLE for DUL

1. create a suitable "init.dul" (See config guide)
2. create a control.dul See above
3. scan the database for segment headers and extents:

```
    $ dul
    UnLoader: Version 2.0.0.0 - Very Restricted on Tue May 16 11:10:16 1995
    Copyright (c) 1994/95 Oracle Corporation, The Netherlands. All rights reserved.
    DUL> scan database;
    data file 1 20480 blocks scanned
    data file 4 7680 blocks scanned
    data file 5 512 blocks scanned
    DUL>quit
```

4. Restart DUL and scan the found tables for column statistics this creates a huge amount of output:

```
echo scan tables \; | dul > scan.out&

[ many lines here]


Object id 1601 table number 0
UNLOAD TABLE T1601_0 ( C1 NUMBER, C2 UNKNOWN, C3 UNKNOWN, C4 NUMBER, C5 DATE
        , C6 NUMBER, C7 NUMBER, C8 NUMBER )
    STORAGE ( TABNO 0 EXTENTS( FILE 1 BLOCK 10530));

Colno  Seen MaxIntSz Null% C75% C100 Num% NiNu% Dat% Rid%
    1    14        3    0%   0%   0% 100% 100%   0%   0%
    2    14        6    0% 100% 100% 100%  14%   0%  21%
    3    14        9    0% 100% 100% 100%  14%   0%   0%
    4    14        3    7%   0%   0% 100% 100%   0%   0%
    5    14        7    0%   0%   0%   0%   0% 100%   0%
    6    14        3    0%   0%   0% 100% 100%   0%   0%
    7    14        2   71%   0%   0% 100% 100%   0%   0%
    8    14        2    0%   0%   0% 100% 100%   0%   0%

"7369" "SMITH" "CLERK" "7902" "17-DEC-1980 AD 00:00:00" "800" "" "20"

"7499" "-0.000025253223" "SALESMAN" "7698" "20-FEB-1981 AD 00:00:00" "1600" "30+

0" "30"

"7521" "WARD" "SALESMAN" "7698" "22-FEB-1981 AD 00:00:00" "1250" "500" "30"

"7566" "JONES" "MANAGER" "7839" "02-APR-1981 AD 00:00:00" "2975" "" "20"

"7654" "MARTIN" "SALESMAN" "7698" "28-SEP-1981 AD 00:00:00" "1250" "1400" "30"

[ many more lines here ]
```

This looks familiar, use the above information and your knowledge of the emp table to compose:

```
UNLOAD TABLE emp ( empno number, ename char, job char, mgr number,
        hiredate date, sal number, comm number deptno number)
 STORAGE ( TABNO 0 EXTENTS( FILE 1 BLOCK 10530));
```

5. use this statement to unload emp:

```
$ dul
UnLoader: Version 2.0.0.0 - Very Restricted on Tue May 16 11:46:33 1995
Copyright (c) 1994/95 Oracle Corporation, The Netherlands. All rights reserved.
Loaded 350 segments
Loaded 204 extents
Extent map sorted
DUL> UNLOAD TABLE emp ( empno number, ename char, job char, mgr number,
DUL 2> hiredate date, sal number, comm number deptno number)
DUL 3> STORAGE ( TABNO 0 EXTENTS( FILE 1 BLOCK 10530));
. unloading table                      EMP      14 rows unloaded
DUL>quit
```

Example unload session: Incorrect init.dul Parameters

WRONG osd_dba_file_bits size

This can generate output similar to below. Normally this should not happen since you should create a demo database and check this via the DUL documented (in html page) query.

 The mismatch in DBA's is only in the file number (first number in brackets) part.
The second number, the block number, is correct.

```
Data UnLoader: Release 3.2.0.1 - Internal Use Only - on Wed Sep 3 10:40:33 1997
Copyright (c) 1994/95 Oracle Corporation, The Netherlands. All rights reserved.
Session altered.
Session altered.
Session altered.
Session altered.
Session altered.
DUL: Warning: Block[1][2] DBA in block mismatch [4][2]
DUL: Warning: Bad cache layer header file#=1, block#=2

DUL: Warning: Block[1][3] DBA in block mismatch [4][3]
DUL: Warning: Bad cache layer header file#=1, block#=3

...........and etc..........
```

WRONG osd_file_leader_size

This may create output similar to below, but many other flavours are possible. In this case we are a fixed number of blocks off. The file number is correct. The difference in the block numbers is constant.:

```
Data UnLoader: Release 3.2.0.1 - Internal Use Only - on Wed Sep 3 10:44:23 1997
Copyright (c) 1994/95 Oracle Corporation, The Netherlands. All rights reserved.
Session altered.
Session altered.
Session altered.
Session altered.
Session altered.

DUL: Warning: Block[1][2] DBA in block mismatch [1][3]
DUL: Warning: Bad cache layer header file#=1, block#=2

DUL: Warning: Block[1][3] DBA in block mismatch [1][4]
DUL: Warning: Bad cache layer header file#=1, block#=3

...........and etc..........
```

WRONG osd_c_struct_alignment

This may generate output similar to the following:

```
Data UnLoader: Release 3.2.0.1 - Internal Use Only - on Wed Sep 3 10:46:10 1997
Copyright (c) 1994/95 Oracle Corporation, The Netherlands. All rights reserved.
Session altered.
Session altered.
Session altered.
Session altered.
Session altered.
. unloading table OBJ$

DUL: Warning: file# 0 is out of range
DUL: Warning: Cannot read data block file#=0, block# = 262145
OS error 2: No such file or directory

DUL: Warning: file# 0 is out of range
DUL: Warning: Cannot read data block file#=0, block# = 262146
OS error 2: No such file or directory

...........and etc..........
```

WRONG db_block_size

```
The following output was generated when the db_block_size was set too
small. The correct value was 4096 and it was set to 2048. Normally, the
value for this parameter should be taken from the Oracle instances's init.ora
file and will not be correctly set.
```

```
Data UnLoader: Release 3.2.0.1 - Internal Use Only - on Thu Sep 4 12:38:25 1997
Copyright (c) 1994/95 Oracle Corporation, The Netherlands. All rights reserved.
Session altered.
Session altered.
Session altered.
Session altered.
Session altered.
DUL: Warning: Block[1][2] DBA in block mismatch [513][1159680]
DUL: Warning: File=1, block 2: illegal block version 2
DUL: Warning: Block[1][2] Illegal block type[0]
DUL: Warning: Bad cache layer header file#=1, block#=2

DUL: Warning: Block[1][4] DBA in block mismatch [1][2]
DUL: Warning: File[1]Block[4]INCSEQ mismatch[90268!=0]
DUL: Warning: Bad cache layer header file#=1, block#=4

DUL: Warning: Block[1][6] DBA in block mismatch [1][3]
DUL: Warning: File[1]Block[6]INCSEQ mismatch[139591710!=86360346]
DUL: Warning: Bad cache layer header file#=1, block#=6

...........and etc..........
```

QUOTE MISSING

```
If you get the following error it is caused by the data dictionary tables
"USER$, OBJ$, TAB$ and COL$" not being correctly generated. To
fix this error simply delete all dictv6.ddl or dictv7.ddl created .dat
and .ctl files and restart.
```

```
Data UnLoader: Release 3.2.0.1 - Internal Use Only - on Wed Sep 3 10:49:30 1997


Copyright (c) 1994/95 Oracle Corporation, The Netherlands. All rights reserved.


DUL: Error: Quote missing
```

Salvaging data from corrupt EXP dump files - UNEXP Tutorial

If you do not know anything about the structure of a EXP dump file this can be difficult. Here is a quick explanation. Apart from the file header a dump file has MARKERS that identify the various sections. In each table section there will be SQL statements. The most interrsesting part is the create table statement, followed by the insert into table statement. The insert statement is directly followed by the bind information, (number of columns, and for each column its type and bind length and a small bit more). Then it is followed by the actual columns. Each column is preceded by a two byte length, followed by the actual column data. There are several tricks for longer columns possible. The end of the column data is marked by the special length marker OXFFFF. There is no marker for the beginning of a row. Resynching after a corruption is trial and error. Corruption are generally not immediate detectable. The format is slightly different for DIRECT export, so you will have to use the DIRECT option for DIRECT exports. The offset to be specified is the beginning of a row. In general the first one directly behind the bind array, but for optimal flexibility you can start anywhere in the row data.

The first step is to scan the dump file to find the offsets and the sql statements. Each output line starts with the offset where the item is found.

```
DUL>  scan dump file expdat.dmp;
0: CSET: 1 (US7ASCII)                 # Character set info from the header
3: SEAL EXPORT:V10.02.01              # the Seal - the exp version tag
20: DBA SYSTEM                        # exp done as SYSTEM
8461: CONNECT SCOTT                   # section for user SCOTT
8475: TABLE "EMP"
                                      # complete create table staement
8487: CREATE TABLE "EMP" ("EMPNO" NUMBER(4, 0), "ENAME" VARCHAR2(10),
"JOB" VARCHAR2(9), "MGR" NUMBER(4, 0), "HIREDATE" DATE, "SAL" NUMBER(7, 2),
"COMM" NUMBER(7, 2), "DEPTNO" NUMBER(2, 0))  PCTFREE 10 PCTUSED 40
INITRANS 1 MAXTRANS 255 STORAGE(INITIAL 65536 FREELISTS 1
FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "USERS" LOGGING NOCOMPRESS
```

```
                                     # Insert statement
8829: INSERT INTO "EMP" ("EMPNO", "ENAME", "JOB", "MGR", "HIREDATE",
"SAL", "COMM", "DEPTNO") VALUES (:1, :2, :3, :4, :5, :6, :7, :8)

                                     # BIND information
8957: BIND information for 8 columns
 col[  1] type 2 max length 22
 col[  2] type 1 max length 10 cset 31 (WE8ISO8859P1) form 1
 col[  3] type 1 max length 9 cset 31 (WE8ISO8859P1) form 1
 col[  4] type 2 max length 22
 col[  5] type 12 max length 7
 col[  6] type 2 max length 22
 col[  7] type 2 max length 22
 col[  8] type 2 max length 22
Conventional export                  # Conventional means NOT DIRECT

9003: start of table data            # Here begins the first row
```

Now build an unexp statement from the create table statement and the direct/conventional information and the start of the column data.

```
UNEXP TABLE "EMP" ("EMPNO" NUMBER(4, 0), "ENAME" VARCHAR2(10),
"JOB" VARCHAR2(9), "MGR" NUMBER(4, 0), "HIREDATE" DATE, "SAL" NUMBER(7, 2),
"COMM" NUMBER(7, 2), "DEPTNO" NUMBER(2, 0))
dump file expdat.dmp from 9003;

Unloaded 14 rows, end of table marker at 9670 # so we have our famous 14 rows


This builds the normal SQL*Loader file and a matching control file.
In the output file one extra column is added, this is related to the
status of the row. A P means the row is Partial, (some columns missing)
R means Resynch, it is the first row after a resynchronisation.
O means Overlap, the previous row had errors, but the new row
partly overlaps the other one.
```

Bernard van Duijnen (Bernard.van.Duijnen@Oracle.com)