1 Introduction 7
2 Prerequisites and conditions 9
2.1 DUDE as a product 9
2.2 DUDE as a service 10
2.3 Getting started 11
2.3.1 Quick inventory 11
2.3.2 Contact us - We Dare We Share We Care – ORA600 partners 12
2.3.3 Probe your environment 13
2.3.4 Start recovering your data 13
2.3.5 Proof of concept 14
3 Configuring DUDE : dude.cfg 15
3.1 Directory parameters 15
3.1.1 LOG_DIR 15
3.1.2 DUMP_DIR 15
3.1.3 DICTIONARY_DIR 16
3.1.4 PLSQL_DIR 16
3.1.5 SCAN_DIR 16
3.1.6 SQLLDR_DIR 17
3.1.7 DDL_DIR 17
3.1.8 SCRIPT_DIR 17
3.1.9 LOB_DIR 17
3.2 Autogeneration parameters 18
3.2.1 GEN_SQLLDR_CTL 18
3.2.2 GEN_TABLE_DDL 18
3.2.3 GEN_SQLLDR_STR_MODE 18
3.3 Sizing parameters 19
3.3.1 BLOCKSIZE 19
3.3.2 BUFFERCACHE 19
3.4 Tablespace and (Data)File parameters 20
3.4.1 LOG_FILE 20
3.4.2 TABLESPACE 20
3.4.2.1 DATAFILE 21
3.4.2.2 OFFSET 21
3.4.2.3 BLOCKSIZE 21
3.4.2.4 ASSM 21
3.4.2.5 SAMPLE_SIZE 21
3.4.2.6 SAMPLE_LIMIT 22
3.4.2.7 BIGFILE 22
3.5 Autocreation parameters 23
3.5.1 AUTO_FILENUMBER 23
3.5.2 AUTO_DICTIONARY 23
3.5.3 AUTO_BLOCK_CHECK 23
3.6 Segment header/extent unloading parameters 25
3.6.1 USE_SEGMENT_HEADER 25
3.6.2 MBRC 25

3.7 Dump parameters 26
3.7.1 EXPORT_MODE 26
3.7.2 EXPORT_FILE_SIZE 26
3.7.3 RECORDLENGTH 27
3.7.4 EXPORT_BUFFER_INCR 27
3.7.5 EXPORT_GZIP 27
3.7.6 GZIP_BUFFER 27
3.7.7 EXPORT_NLS 28
3.7.8 FLAT_NLS 28
3.7.9 FIELD_ENCL 28
3.7.10 FIELD_SEP 28
3.7.11 RECORD_SEP 29
3.7.12 NULL_FIELD 29
3.7.13 LOBS_ALWAYS_SQLLDR 29
3.7.14 HUGE_LOB_SUPPORT 29
3.7.15 PRETTY_FILE_NAMES 30
3.7.16 INCLUDE_DROPPED 30
3.7.17 DELETED_ONLY 31
3.8 Cross platform unloading parameters 32
3.8.1 MAP_DUMP_DIR 32
3.8.2 MAP_SQLLDR_DIR 32
3.8.3 MAP_LOB_DIR 32
3.8.4 MAP_FILESEP 33
3.9 Transparent Data Encryption – TDE – parameters 34
3.9.1 ENABLE_TDE 34
3.9.2 TDE_WALLET 34
3.9.3 TDE_WALLET_PASSWORD 34
3.10 Fractured blocks 35
3.10.1 FLAG_FRACTURED_BLOCKS 35
3.10.2 SKIP_FRACTURED_BLOCKS 35
3.10.3 SKIP_OUT_OF_SYNC 35
3.11 Various parameters 36
3.11.1 VERSION 36
3.11.2 PLATFORM 36
3.11.3 PARTITIONING 36
3.11.4 ENABLE_LOBS 36
3.11.5 NO_FILESEP 37
3.11.6 INCLUDE_BLOCK_DUMP 37
3.12 Bootstrap parameters 38
3.12.1 BOOTSTRAP_OBJ_OID 38
3.12.2 BOOTSTRAP_COBJ_OID 38
3.12.3 BOOTSTRAP_CUSER_OID 38
3.12.4 BOOTSTRAP_TAB_TABNO 39
3.12.5 BOOTSTRAP_COL_TABNO 39
3.12.6 BOOTSTRAP_LOB_TABNO 39

3.12.7 BOOTSTRAP_USER_TABNO 40
3.12.8 BOOTSTRAP_CFILEBLOCK_OID 40
3.12.9 BOOTSTRAP_SOURCE_OID 40
3.13 Deprecated parameters 42
3.13.1 BIG_ENDIAN 42
3.14 Hidden parameters 43
3.14.1 _SKIP_BLOB_TABLE 43
3.14.2 _FILENUMBER_BLOCK 43
3.15 Example dude.cfg 44
4 Getting started : DUDE 46
4.1 Java version 46
4.2 Running DUDE 46
4.3 Statements 47
5 Blockmaps 48
5.1 CREATE BLOCKMAP FOR TABLESPACE statement 49
5.2 DROP BLOCKMAP FOR TABLESPACE statement 50
6 Extent maps 51
6.1 Pro and contra 51
6.2 SYSTEM tablespace considerations 51
6.3 Extentmap configuration 52
7 Do you speak Oracle – the dictionary 53
7.1 DUMP DICTIONARY 53
7.2 CREATE DICTIONARY 54
7.3 CROSSCHECK DICTIONARY 54
8 Getting information out of your dictionary 56
8.1 SHOW OBJECTID <objectid> 56
8.2 SHOW TABLE <username> <tablename> 57
8.3 SHOW USERS 58
8.4 SHOW TABLESPACES 58
8.5 SHOW TABLESPACE <tablespace name> 58
8.6 SHOW TABLESPACES FROM DICTIONARY 58
8.7 SHOW HEADER <objectid> 60
9 Did you say Oracle 6/7 – the migrated dictionary 61
9.1 How do we know this was once an Oracle 7 beauty 61
9.2 SCAN BOOTSTRAP 65
10 Dumping data 70
10.1 DUMP OBJECTID 70
10.2 DUMP OBJECTID <table objectid> RESUME AFTER PARTITION <partition objectid> 71
10.3 DUMP TABLESPACE 71
10.4 DUMP TABLESPACE <tablespace_name> RESUME AFTER <objectid> 73
10.5 DUMP USER 74
10.6 DUMP USER <user> RESUME AFTER <objectid> 74

10.7 DUMP USER <user> RESUME AFTER <table objectid> PARTITION <partition objectid> ... 74
10.8 DUMP TABLE ... 74
10.9 Empty tables ... 75
10.10 Table types ... 75
11 Generating DDL ... 76
11.1 GENERATE DDL FOR ... 76
11.2 GENERATE INDEX DDL ... 76
11.3 GENERATE SQUENCE DDL ... 76
11.4 GENERATE VIEW DDL ... 77
11.5 Dumping PLSQL code ... 77
12 No SYSTEM, no problem ... 79
12.1 Is SYSTEM really gone ? ... 79
12.2 Getting started ... 80
12.2.1 Blockmaps ... 80
12.2.2 Scan data ... 80
12.2.2.1 SCAN OBJECTID ... 80
12.2.2.2 SCAN TABLESPACE ... 81
12.2.2.3 SCAN TABLESPACE ORPHANED ... 83
12.2.3 Extracting scanned data ... 83
12.2.3.1 DUMP SCANNED OBJECTID ... 83
12.2.3.2 DUMP SCANNED TABLESPACE ... 84
12.3 Scanning issues and limitations ... 85
12.3.1 Dropped tables ... 85
12.3.2 Non-heap tables ... 85
12.3.3 Trailing NULL columns ... 85
12.3.4 Wrong datatype ... 86
12.3.5 Supported datatypes ... 87
12.3.6 Missing tables ... 88
12.3.7 Internal column order ... 88
13 Miscellaneous ... 89
13.1 START <dude script> command ... 89
14 Addendum A : Valid values for EXPORT_NLS ... 91
15 Addendum B : Valid values for FLAT_NLS ... 93
15.1 Basic Encoding Set (contained in rt.jar) ... 93
15.2 Extended Encoding Set (contained in i18n.jar) ... 93
16 Addendum C : Quick start – SYSTEM available ... 99
16.1 Generate a probe file ... 99
16.1.1 Generate dude_probe.cfg ... 99
16.1.2 Run DUDE_PROBE.jar ... 99
16.2 Generate dude.cfg file ... 100
16.3 Startup DUDE ... 101
16.4 Create blockmaps for all tablespaces ... 102
16.5 Dump the dictionary ... 103
16.6 Create the dictionary in DUDE ... 104

16.7 Test the dictionary by list the users 104
16.8 Dump the data 105
17 Addendum D : Quick start – SYSTEM not available 106
17.1 Generate a probe file 106
17.1.1 Generate dude_probe.cfg 106
17.1.2 Run DUDE_PROBE.jar 106
17.2 Generate dude.cfg file 107
17.3 Startup DUDE 108
17.4 Create blockmaps for all tablespaces 109
17.5 Scan the tablespace for segments 109
17.6 Dump the scanned tables/tablespaces 112
17.7 Check dumped data 113
17.8 Identify the data 113
17.9 Load the data 114
18 Troubleshooting 115
18.1 java.lang.OutOfMemoryError exception (1) 115
18.2 java.lang.OutOfMemoryError exception (2) 115
18.3 DUMP DICTIONARY throws lots of errors 115
18.3.1 ASCII mode FTP 115
18.3.2 Wrong PLATFORM parameter 116
18.4 All tables return with 0 records while dumping 117
18.5 DUDE tries to load a block from a ridicules blocknumber 117
18.6 Combination of all the above problems 118
19 Acknowledgements 119

Roads ?
where we're going, we don't need roads.
- Doc, Back To The Future.

1 Introduction

This document contains instructions for using DUDE (Database Unloading by Data Extraction). DUDE was also known as jDUL. jDUL refers to Oracle's DUL (Data Unloader). DUDE offers the same functionality as DUL in a multi-platform java environment.
Basically, DUDE is able to recover data from a crashed and/or corrupted database. However, it should only be used as a last resort!
DUDE will also extract uncommited data – in other words, dirty reads are done.

DUDE v2.8.4 currently supports :

- creation of blockmaps
- extraction of extentmaps
- recreation of the Oracle dictionary
- autodetection of file number
- performing a describe of tables
- extracting tables based on object id or tablename
- extracting entire tablespaces
- extracting partitioned tables (including subpartitions)
- extracting PLSQL code
- generating table DDL
- generating sequence DDL
- generating view DDL
- generating index DDL
- heuristic scanning : in case of missing SYSTEM tablespace – tables, columns, and datatypes are guessed through sampling of datasets
- data can be extracted to the following output formats :
    - sql*loader flatfiles – autogeneration of sql*loader controlfiles
    - Oracle 8.0.5 compatible exp DMP files
        - mutiple DMP file support (for filesystems with 2Gb limits)
        - DMP files can be compressed on the fly in gzip format
        - the 8.0.5 DMP's can be imported in 8.0.x, 8.1.x, 9.x.x and 10.x.x databases
- all major intel & unix platforms are supported : windows, linux, IBM AIX, Sun Solaris/SunOS, HPUX, Tru64 & VMS
- datatypes supported : NUMBER, CHAR, VARCHAR2, LONG, LONG RAW, RAW, DATE, BLOB, CLOB, TIMESTAMP (+9i); BINARY FLOAT, BINARY DOUBLE (+10g)
- LOB support :
    - CLOB & BLOB support :
        - support for *enable storage in row inline* LOBs

        - support for *enable storage in row out-of-line* LOBs
        - support for *disable storage in row* LOBs
        - support for chunk versioning
    - LOBs are only supported when a SYSTEM tablespace is available
- Indexed Organized Tables – IOT support for :
    - Index only IOT's
    - IOT's with overflow segments
    - Partitioned index only IOT's
    - Partitioned IOT's with overflow segments
    - SYSTEM tablespace needs to be available for IOT support
- supports Oracle RDBMS versions 7, 8.0, 8i, 9i, 10g and 11g
- supports Oracle 10g BIGFILE type tablespaces (see metalink note 262472.1)
- cross-platform unloading
- chained and migrated rows (this includes intra-row chaining), trailing NULL columns, first free row index
- support for migrated dictionaries (for example Oracle 7 dictionary migrated to 8.x using mig tool) using bootstrap procedure
- support for compressed blocks
- support for 11g table compression (COMPRESS FOR ALL OPERATIONS) (currently in beta)
- detection of physical block corruption and fractured blocks
    - possibility to flag the block and/or skip the complete block or skip part of the block

DUDE v2.8.2 currently does not support :

- TDE
- 11g securefiles
- Oracle ASM
- Oracle 6 - never

2 Prerequisites and conditions

DUDE is not an alternative for Oracle exp or SQL*Plus utilities. You should only use DUDE when :

- you're unable to open the database using standard restore and recovery techniques
- you're unable to open the database using non-standard recovery techniques (the use of hidden Oracle parameters - _allow_..._corruption)
- Oracle Support tells you the database is death, gone, kapot, cassé, tod, сломанный, roto - but data can be recovered using a DUL-like tool
- you accept the fact that there's **NO** guarantee – there's a reason why the database doesn't open anymore ! (this is true for *all* DUL-like tool)

2.1 DUDE as a product

DUDE can be leased for a period of time (default 7 days). This means *you* get to run the program on *your* server. DUDE is designed to specifically run **on one (and only one)** server and **one (and only one)** database.
Prerequisites :

- Java runtime should be at least 1.4.x : to check, run *java –version* or *jre –version* (always let us know which java version you're on)
- run DUDE_PROBE.jar (see http://www.ora600.be) and send us the output
- a senior DBA operating DUDE, assisted by us
- in case of a missing SYSTEM tablespace : a *senior* application developer to identify all data (mapping to tables and columns)

If for some reason you're unable to provide the correct Java runtime on the server (for example with older systems), you can always place the datafiles on a different machine. DUDE supports cross platform unloading. So you can easily ftp those datafiles from your old and dusty unix box to your brand-new portable and run DUDE from there.

After the lease period, DUDE becomes useless.
This option is ideal if your database contains sensitive data (credit card numbers, patient files) that should not leave the server.

Pricing : fixed price – includes support

2.2 DUDE as a service

We run DUDE on :

- *your* server
  - a VPN connection should be provided to allow access to your database server
  - Java runtime should be at least 1.4.x

- *our* server
  - datafiles should be uploaded to our server

- on-site consultancy
  - provided by NRG Consultancy www.nrg.co.za in South Africa
  - ask your local ORA600 partner

In case of a missing SYSTEM tablespace it's in your best interest that you buy DUDE as a product. We have no affinity with your data, so a lot of time may be lost identifying the correct column datatypes.

Data is ***EXTRACTED ONLY!*** If you want further assistance loading the extracted data in a new database, our partners will gladly help you out.

Pricing : hourly rate (min 4hours)

2.3 Getting started

2.3.1 Quick inventory

Before contacting us, please make sure you are able to provide us with following information :

- database size ?
- database version ?
- platform the database ran on ?
- is SYSTEM lost ?
- preferred deployment type :
    - will you upload the datafiles ?
    - will you provide us with a connection to your server (or other machine) containing the datafiles ?
    - will you do the extraction yourself ?

The more info you can provide us, the better we can help you.

2.3.2 Contact us - We Dare We Share We Care – ORA600 partners

Europe

- ORA600 bvba, Belgium – www.ora600.be
  Please contact Kurt Van Meerbeeck, Cell: +32 495 580 714 or dude@ora600.be for further details.

- Miracle AS, Denmark – www.miracleas.dk.
  Please contact Henrik Bjerknæs Rasmussen, Cell: +45 25 277 110 for further details or hra@miracleas.dk

North America

- Evergreen Database Technologies Inc., USA – www.evdbt.com
  Please contact Tim Gorman, Cell: +1 303 885 4526 or tim@evdbt.com or dude@evdbt.com

- Optimal DBA, USA – www.optimaldba.com
  Please contact Daniel Fink, Cell : +1 303 808 32 82 or Daniel.fink@optimaldba.com or dude@evdbt.com

- Pythian, Canada - http://www.pythian.com/contact
  Please contact Alex Gorbachev – email gorbachev@pythian.com or dude@pythian.com

Africa

- NRG Consulting, South Africa - www.nrgc.co.za
  Please contact Kugendran Naidoo, Cell: +27 82 7799275 or k@nrgc.co.za

South America

- HBTec, Brazil – www.hbtec.com.br
  Please contact Jefferson Luiz Prebianca, Cell: +55 47 88497639 or dude@hbtec.com.br

East Asia Pacific

- Pythian, Australia - http://www.pythian.com/contact
  Please contact Alex Gorbachev – email gorbachev@pythian.com or dude@pythian.com

For all technical inquiries contact Kurt Van Meerbeeck, Cell: +32 495 580 714 or dude@ora600.be

2.3.3 Probe your environment

If you or your team will do the extraction yourself – you'll have to run DUDE_PROBE.jar first. This tool will probe the machine where DUDE will later run on.

DUDE CAN **ONLY RUN** ON YOUR MACHINE **AFTER DUDE_PROBE** HAS **PROBED** IT **!**

See http://www.ora600.be for more info on running DUDE_PROBE.jar

2.3.4 Start recovering your data

Once your environment has been probed – DUDE can be deployed.

2.3.5 Proof of concept

We can provide you with a proof of concept to convince you of DUDE's capabilities. This can be done by either :

- uploading of datafiles : SYSTEM tablespace (if available) + one other datafile
- we can give you a limited version of DUDE that will only extract the **first 10 rows of the first block** of a table. For this we need a probe.out file (see 2.3.3 Probe your environment)

3 Configuring DUDE : dude.cfg

Before getting started you will need to adjust the DUDE configuration file : *dude.cfg*.
Always place dude.cfg in the same directory as *dude.class*.
dude.cfg contains all parameters necessary to run DUDE.
All parameter values must be enclosed by double quotes.
DUDE supports Java-like comments in *dude.cfg* (// and /* ... */).

3.1 Directory parameters

The following parameters all specify directories where files will be placed. Make sure all directories exist and are writeable by DUDE.
For windows user – you must escape the backward slash. For example :
LOG_DIR="d:\\dude\\log_dir"
(L)Unix users – use normal forward slash. Example : *LOG_DIR="/dude/log_dir"*

3.1.1 LOG_DIR

default value = <not defined>
type = mandatory
example : LOG_DIR="c:\\dude\\log_dir"

- specifies the directory in which the log file (*LOG_FILE*) will be placed.

3.1.2 DUMP_DIR

default value = <not defined>
type = mandatory
example : DUMP_DIR="c:\\dude\\dump_dir"

- specifies the directory in which all data dumps will be placed. All data dumps have extension *.dat and are flatfiles to be used with the Oracle's sql*loader utility.
- for every *.dat file you'll find a matching sql*loader control file in *SQLLDR_DIR*
- for performance reasons you should place *DUMP_DIR* on a seperate IO device (separate from the datafiles to be extracted)

3.1.3 DICTIONARY_DIR

default value = <not defined>
type = mandatory
example : DICTIONARY_DIR="c:\\dude\\dictionary_dir"

- specifies the directory in which DUDE places :
  - all blockmaps :
    - one file per OBJECTID containing block information
    - one file per TABLESPACE containing OBJECTID information
  - dictionary tables :
    - obj.dat : contains dump of OBJ$
    - user.dat : contains dump of USER$
    - col.dat : contains dump of COL$
    - tab.dat : contains dump TAB$
    - tabpart.dat : contains dump of TABPART$ *(optional - if partitioning is enabled)*
    - PLSQL.dat : contains dump of source$ *(optional- in case of dump PLSQL code)*

3.1.4 PLSQL_DIR

default value = <not defined>
type = optional
example : PLSQL_DIR="c:\\dude\\plsql_dir"

- specifies the directory where all *sorted* plsql code is placed

3.1.5 SCAN_DIR

default value = <not defined>
type = optional
example : SCAN_DIR="c:\\dude\\scan_dir"

- specifies the directory where all DDE files are placed. A *.dde file contains DUDE commands to unload scanned tables
- A scanned table is a table which was discovered by DUDE's heuristic scan routine.
- *SCAN_DIR* should only be used if the SYSTEM tablespace is not available.

3.1.6 SQLLDR_DIR

default value = <not defined>
type = optional
example : SQLLDR_DIR="c:\\dude\\sqlldr_dir"

- specifies the directory where all sql*loader control files (*.ctl) are placed
- see *3.2.1 GEN_SQLLDR_CTL* parameter

3.1.7 DDL_DIR

default value = <not defined>
type = optional
example : SQLLDR_DIR="c:\\dude\\ddl_dir"

- specifies the directory where all DDL files (*.sql) are placed
- see *3.2.2 GEN_TABLE_DDL* parameter
- see *11 Generating DDL*

3.1.8 SCRIPT_DIR

default value = current directory
type = optional
example : SCRIPT_DIR="c:\\dude\\script_dir"

- specifies the directory where custom dude scripts reside
- see also the *13.1 START <dude script> command*

3.1.9 LOB_DIR

default value = <not defined>
type = optional
example : LOB_DIR="c:\\dude\\lob_dir"

- specifies the directory where BLOB files will reside
- only valid when *ENABLE_LOBS* is *true* (cfr. *3.11.4 ENABLE_LOBS*)

3.2 Autogeneration parameters

3.2.1 GEN_SQLLDR_CTL

default value = true
type = optional
example : GEN_SQLLDR_CTL="true"

- specifies if sql*loader control files are generated when unloading data
- the control files are placed in the *SQLLDR_DIR* directory

3.2.2 GEN_TABLE_DDL

default value = true
type = optional
example : GEN_TABLE_DDL="true"

- specifies if table DDL (data definition language) scripts are generated when unloading data
- the DDL scripts are placed in the *DDL_DIR* directory

3.2.3 GEN_SQLLDR_STR_MODE

default value = true
type = optional
example : GEN_SQLLDR_STR_MODE="true"

- starting from sql*loader version 8.1.6, stream (str) mode is supported to load embedded newlines
- if you are going to load the generated flatfiles into a 8.1.6 or higher DB – set this parameter to true – otherwise, set it to false
- if set to false, you should set *RECORD_SEP* accordingly :
    - on Unix use '\n' as an end of line marker – on windows use '\r\n'
- see also *3.7.11 RECORD_SEP*

3.3 Sizing parameters

The sizing parameters define block and memory structures.

3.3.1 BLOCKSIZE

default value = <not defined>
type = mandatory
example : BLOCKSIZE="2048"

- defines the size of the Oracle database block. This parameter is inherited by each defined tablespace (see *3.4.2 TABLESPACE*). Although every value is accepted, valid values should be 2048, 4096, 8192, 16384 and 32768. *BLOCKSIZE* should reflect the database's BLOCKSIZE parameter

3.3.2 BUFFERCACHE

default value = "1000"
type = mandatory
example : BUFFERCACHE="10000"

- defines the size of DUDE's internal buffercache. The unit of *BUFFERCACHE* is *BLOCKSIZE*. Every read/write thread will allocated *BLOCKSIZE*BUFFERCACHE* of memory.
- at the moment, only one concurrent dump operation is supported. However, future releases will support multiple concurrent dump operations – thus, every concurrent dump operation will allocate its own buffercache.

3.4 Tablespace and (Data)File parameters

3.4.1 LOG_FILE

default value = <not defined>
type = mandatory
example : LOG_FILE="dude.log"

- defines the name of DUDE's logfile
- DUDE automatically rotates its logfile every 10Mb
- A second, more compact logfile will be generated for each run, with the name *compact_<LOG_FILE>*
- The *compact_<LOG_FILE>* shows row statistics for each table/partition/iot on one line
- Statistics explanation :
    - NR = amount of normal rows unloaded
    - MR = amount of migrated rows unloaded
    - CR = amount of chained rows unloaded
    - DR = amount of deleted rows found – these are not unloaded as they are marked as deleted
    - SR = amount of rows skipped due to corruption or errors
    - Total = NR+MR+CR

3.4.2 TABLESPACE

default value = <not defined>
type = mandatory
example :

```
TABLESPACE "SYSTEM"
{
        DATAFILE="d:\\oradata\\orcl\\system01.dbf"
        DATAFILE="d:\\oradata\\orcl\\system02.dbf"

}

TABLESPACE "DUDE"
{
        DATAFILE="d:\\oradata\\orcl\\dude01.dbf"
        ASSM="false"
        BIGFILE="true"
        BLOCKSIZE="8192"
}
```

3.4.2.1 DATAFILE

default value = <not defined>
type = mandatory

- specifies the datafile to be extracted

3.4.2.2 OFFSET

default value = "0"
type = optional

- if defined, DUDE will skip the first <<*OFFSET*>> bytes from the datafile
- should only be used for raw devices on certain platforms (for example AIX)

3.4.2.3 BLOCKSIZE

default value = <INSTANCE BLOCKSIZE>
type = optional

- specifies the tablespace specific blocksize
- if not defined within the *TABLESPACE* clause, the tablespace inherits the database blocksize (cfr *2.2.1 BLOCKSIZE*)
- starting from Oracle 9i, tablespaces can be created with a blocksize different from the database blocksize
- valid values (cfr *2.2.1 BLOCKSIZE*)

3.4.2.4 ASSM

default value = "false"
type = optional

- Oracle 9i introduced automatic segment storage management
- set this parameter to *"true"* when the tablespace is created using ASSM

3.4.2.5 SAMPLE_SIZE

default value = "100"
type = optional

- defines the percentage of blocks to scan for the heuristic scan routine

- a value of 100 will scan all (100%) blocks of an object – a value of 10 will scan only 10% of all blocks
- basically it defines the dataset to be sampled for guessing columns and datatypes when the SYSTEM tablespace is not available
- only to be used when the SYSTEM tablespace (read dictionary) is not available

3.4.2.6 SAMPLE_LIMIT

default value = <not defined>
type = optional

- defines the absolute number of blocks to scan for the heuristic scan routine
- contrary to SAMPLE_SIZE, this parameter limits the dataset by using an absolute number of blocks
- *SAMPLE_SIZE* and *SAMPLE_LIMIT* can be combined
- only to be used when the SYSTEM tablespace (read dictionary) is not available

example :
```
TABLESPACE "DUDE"
{
        DATAFILE="d:\\oradata\\orcl\\dude01.dbf"
        ASSM="false"
        BLOCKSIZE="8192"
        SAMPLE_SIZE="50"
        SAMPLE_LIMIT="1000"
}
```

3.4.2.7 BIGFILE

default value = "false"
type = optional

- Oracle 10g introduced BIGFILE tablespaces – these are tablespaces consisting of one and only datafile that can have 2^{32} blocks.
- set this parameter to *"true"* when the tablespace is created as a BIGFILE tablespace

3.5 Autocreation parameters

Autocreation parameters trigger commands when DUDE is started. It becomes quite frustrating to perform certain operations every time DUDE is started. That's where the following two parameters are a lifesaver.

3.5.1 AUTO_FILENUMBER

default value = "true"
type = optional
example : AUTO_FILENUMBER="true"

- if set to *true*, a file number map is automatically created at startup
- you should set this parameter **always** to ***true***
- the command *CREATE FILENUMBERMAP* can still be issued to manually create a file number mapping. However, with this parameter, the command became sort or less obsolete, and therefore it is not discussed in detail in this document.

3.5.2 AUTO_DICTIONARY

default value = "false"
type = optional
example : AUTO_DICTIONARY="true"

- if set to *true*, the dictionary is automatically created at startup
- you should only set this parameter to *true* after you've dumped the Oracle dictionary – otherwise, DUDE will exit with a failure.

3.5.3 AUTO_BLOCK_CHECK

default value = "false"
type = optional
example : AUTO_BLOCK_CHECK="true"

- if set to *true*, the datafile's internal blocksize is checked against the *BLOCKSIZE* parameter
- **<note> this parameter is currently only working correctly on intel and IBM AIX platforms**

3.6 Segment header/extent unloading parameters

DUDE can use Oracle's own extent information to unload data – in this case it will use extra dictionary information and the segment's extent information stored in the segment headers and extent map blocks.
Using this information significantly increases data unloading.
However, there's always the risk that the extent information itself is corrupt. If this is the case, you should use DUDE's blockmap functionality to avoid missing part of the segments data.

3.6.1 USE_SEGMENT_HEADER

default value = "false"
type = optional
example : USE_SEGMENT_HEADER ="true"

This parameters indicates that DUDE should use extent map information as opposed to it's own blockmap capabilities.
This parameter should be set before dumping the dictionary.

WARNINGS – set this parameter to *"false"*

- **if you're seeing a lot of fractured blocks**
- **if you want to unload truncated or dropped segments**
- **if you've lost the SYSTEM tablespace**

3.6.2 MBRC

default value = "128"
type = optional
example : MBRC="128"

This parameter is the equivalent of Oracle's db_file_multiblock_read_count parameter. It defines the average amount of blocks DUDE reads in its cache per IO call.

3.7 Dump parameters

The dump parameters specify the format the extracted data.

3.7.1 EXPORT_MODE

default value =" false"
type = optional
example : EXPORT_MODE = "true"

- specifies DUDE's output mode
- if set to true, Oracle exp DMP compatible files are generated in Oracle 8.0.5 format
- if set to false, ASCII flatfiles are generated

3.7.2 EXPORT_FILE_SIZE

default value = 2097152000
type = optional
example : EXPORT_FILE_SIZE = "2097152000"

- specifies the maximum size (in bytes) of the generated EXP DMP
- needed for situations where there's 2Gb file limit
- this is not the same as Oracle exp FILE_SIZE parameter introduced in Oracle 8i – DUDE generates totally independent EXP DMP files, each with their own DDL, and thus should be imported separately with the IGNORE=Y parameter. The FILE_SIZE should not be used for importing !
- if *EXPORT_GZIP="true"* a new file is created when the amount of **uncompressed** bytes has reached *EXPORT_FILE_SIZE* – **not** when the **compressed file** has reached *EXPORT_FILE_SIZE.*
- example :
    - if a single dump file, called *OWNER_TABLE_ID.dmp*, would turn out to be 5GB and *EXPORT_FILE_SIZE="2097152000"* then the following files will be generated :
        - OWNER_TABLE_ID.dmp : max 2Gb
        - OWNER_TABLE_ID_1.dmp : max 2Gb
        - OWNER_TABLE_ID_2.dmp : max 2Gb
    - to import these files, use Oracle's imp utility **without** the use of the FILE_SIZE parameter – in other words, 3 independent imports :
        - imp system/password FILE=OWNER_TABLE_ID.dmp IGNORE=Y
        - imp system/password FILE=OWNER_TABLE_ID_1.dmp IGNORE=Y

- imp system/password FILE=OWNER_TABLE_ID_2.dmp IGNORE=Y

3.7.3 RECORDLENGTH

default value = 65536
type = optional
example : RECORDLENGHT = "65536"

- specifies the initial exp API record buffer size in bytes
- all row pieces of a record are copied into the recordbuffer when dumping to DMP format
- if corruption is encountered, the buffer is invalidated and the content will not be flushed to disk

3.7.4 EXPORT_BUFFER_INCR

default value = 10485760
type = optional
example : EXPORT_BUFFER_INCR = "10485760"

- Large records that do not fit the record buffer, may extend the buffer
- The record buffer is extended by *EXPORT_BUFFER_INCR* bytes each time it gets too small

3.7.5 EXPORT_GZIP

default value = false
type = optional
example : EXPORT_GZIP = "true"

- specifies if gzip compression is used on the generated EXP DMP files
- compression is done on-the-fly, which potentially could save a lot of disk space
- more CPU is used, but less IO writes are done

3.7.6 GZIP_BUFFER

default value = 131072
type = optional
example : GZIP_BUFFER = "5242880"

- specifies the size (in bytes) of the output buffer used by the GZIP algorithm

- default 128Kb

3.7.7 EXPORT_NLS

default value = US7ASCII
type = optional
example : EXPORT_NLS =“WE8ISO8859P15"

- specifies the database characterset
- only to be used when unloading to exp dmp files
- the export dump created by DUDE is the same as a DIRECT=Y export – this means that the sql engine is bypassed by the exp utility. And as a result, no characterset translation is done. Therefore, *EXPORT_NLS* should reflect *database* characterset (as opposed to the *client's* characterset).
- *See also 14 Addendum A : Valid values for EXPORT_NLS*

3.7.8 FLAT_NLS

default value = ISO8859_1
type = optional
example : FLAT_NLS =“ISO8859_1"

- specifies the Java character encoding scheme for producing flat files
- see *15 Addendum B : Valid values for FLAT_NLS*

3.7.9 FIELD_ENCL

default value = “34”
type = optional
example : FIELD_ENCL=”34”

- defines the field enclosure character as a *DECIMAL (note : not as a HEXADECIMAL!)*
- decimal# character 34 is a double quote (“)

3.7.10 FIELD_SEP

default value = “44”
type = optional
example : FIELD_SEP=”44”

- defines the field separator character as a DECIMAL

- decimal# character 44 is a comma (,)

3.7.11 RECORD_SEP

default value = "\n"
type = optional
example : RECORD_SEP="|||\n"

- defines the record separator as a string
- should be used when carriage return/newline characters are embedded within fields
- starting from sql*loader version 8.1.6, stream (str) mode is supported to load embedded newlines
- important – Unix uses '\n' as an end of line marker – windows uses '\r\n'

3.7.12 NULL_FIELD

default value = "NULL"
type = optional
example : NULL_FIELD=""

- defines how a NULL field should be represented in the dump flatfiles
- DUDE internally uses "NULL", however for sql*loader you should set this to an empty string ("")

3.7.13 LOBS_ALWAYS_SQLLDR

default value = "false"
type = optional
example : LOBS_ALWAYS_SQLLDR="true"

- If set to *TRUE,* and *EXPORT_MODE="true"*, then a table with LOB columns will always be dump in flatfile format

3.7.14 HUGE_LOB_SUPPORT

default value = "false"
type = optional
example : HUGE_LOB_SUPPORT="true"

- If set to *TRUE,* DUDE will use an on-disk-hashing algorithm for identifying lob chunks.
- *HUGE_LOB_SUPPORT* can only be used in combination with *EXPORT_MODE="false"* (flatfile mode)
- When the recordsize is larger than 1Gb you should set *HUGE_LOB_SUPPORT="true"*.

3.7.15 PRETTY_FILE_NAMES

default value = "false"
type = optional
example : PRETTY_FILE_NAMES ="true"

- If set to *TRUE*, tables the dollar ("$") sign character in filenames will be replaced an underscore ("_") character
- On Unix platforms it is quite annoying to always escape the $ sign when using sqlldr and/or exp/imp tools – hence this parameter

3.7.16 INCLUDE_DROPPED

default value = "false"
type = optional
example : INCLUDE_DROPPED ="true"

- If set to *TRUE*, the *dump dictionary* command will unload records that have marked as *deleted*
- A DDL *DROP* command basically *delete*s records within the base dictionary tables. The records will be marked as deleted in the row header. They will be removed once Oracle decides to coalesce the free space within the block.
- It is thus possible, for a limited period of time, to extract 'dropped data' – this is also valid for *dropped PLSQL code* and *sequences*.
- If the delete records can not be extracted from the base dictionary tables, dropped data can still be recovered using the heuristic scan routine using the commands :
    - *SCAN OBJECTID* (see *12.2.2.1 SCAN OBJECTID*)
    - *SCAN TABLESPACE* (see *12.2.2.2 SCAN TABLESPACE*)
    - *SCAN TABLESPACE ORPHANED* (see *12.2.2.3 SCAN TABLESPACE ORPHANED*)

3.7.17 DELETED_ONLY

default value = "false"
type = optional
example : DELETED_ONLY ="true"

- If set to *TRUE*, dump commands will ***only*** unload deleted records
- This can be useful when a delete and commit has occurred on data, and the purpose is to recover only the deleted data. All records that are flagged as deleted are unloaded. Of course, there is no guarantee one will be able to recover all deleted records.

3.8 Cross platform unloading parameters

The following parameters are extremely useful when performing a cross platform unload - for example, while unloading datafiles from a Unix platform on a windows machine.
If you're unloading to flatfiles, all references within the sqlldr controlfile will have Windows style paths. This is a problem when you want to load the data on the original unix machine.
The following parameters will aid you in this by letting you define logical directory values which will be used in favor of the physical directory values.

3.8.1 MAP_DUMP_DIR

default value = DUMP_DIR
type = optional
example :
DUMP_DIR="c:\dump"
MAP_DUMP_DIR="/dump"

- This parameter will replace *DUMP_DIR* within the sqlldr controlfile
- It defines the directory where sqlldr can find it's flatfile for loading

3.8.2 MAP_SQLLDR_DIR

default value = SQLLDR_DIR
type = optional
example :
SQLLDR_DIR="c:\controlfile"
MAP_SQLLDR_DIR="/controlfile"

- This parameter will replace *SQLLDR_DIR* within the sqlldr controlfile
- It defines the directory where sqlldr will place it's *BAD* and *LOG* files

3.8.3 MAP_LOB_DIR

default value = LOB_DIR
type = optional
example :

LOB_DIR="c:\lobs"
MAP_LOB_DIR="/lobs"

- This parameter will replace *LOB_DIR* within the sqlldr flatfile (!)
- It defines the directory where sqlldr can find a record's LOB file within the flatfile

3.8.4 MAP_FILESEP

default value = none
type = optional
example :
MAP_FILESEP="/"

- This parameter will replace you local OS file separator with your target's OS file separator
- The local OS is the OS of the machine your performing the extraction on – for example a MS Windows machine with file separator "\".
- The target OS is the OS of the machine where you will load your data on – for example an IBM AIX machine with file separator "/".

3.9 Transparent Data Encryption – TDE – parameters

TDE data can be extracted and decrypted as long as the master key in the database wallet and the column keys in the dictionary are still intact and available.

Disclaimer – TDE support is not available in the current release

3.9.1 ENABLE_TDE

default value = "false"
type = optional
example : ENABLE_TDE="true"

- When set to *true*, DUDE will support the unloading encrypted data using the TDE database option
- This will result in opening the database keystore during initialization, and such a keystore/wallet password is needed

3.9.2 TDE_WALLET

default value = ""
type = mandatory if ENABLE_TDE="true"
example : TDE_WALLET="/home/app/oracle/admin/wallet/ewallet.p12"

- Absolute path and filename of the database wallet
- The wallet is PKCS12 keystore containing the master key

3.9.3 TDE_WALLET_PASSWORD

default value = ""
type = mandatory if ENABLE_TDE="true"
example : TDE_WALLET_PASSWORD="mywalletpassword"

- The password for opening the database keystore

3.10 Fractured blocks

3.10.1 FLAG_FRACTURED_BLOCKS

default value = "false"
type = optional
example : FLAG_FRACTURED_BLOCKS="true"

- When set to *true*, DUDE will calculate the block checksum and compare it with the block tail. If there's a mismatch, and error message is shown.
- This parameter was introduced to identify fractured blocks where DUDE would throw exceptions when the row directory points to invalid row headers. In other words, the row directory is out-of-sync with the actual data stored in the block.
- This parameter can not be set when *VERSION="7"*

3.10.2 SKIP_FRACTURED_BLOCKS

default value = "false"
type = optional
example : SKIP_FRACTURED_BLOCKS="true"

- When set to *true*, DUDE will calculate the block checksum and compare it with the block tail. If there's a mismatch, the *complete* block is skipped !!!
- This parameter was introduced to identify fractured blocks where DUDE would throw exceptions when the row directory points to invalid row headers. In other words, the row directory is out-of-sync with the actual data stored in the block.
- This parameter can not be set when *VERSION="7"*

3.10.3 SKIP_OUT_OF_SYNC

default value = "false"
type = optional
example : SKIP_OUT_OF_SYNC="true"

- It is possible that in some specific cases (when blocks are fractured) the row directory is out-of-sync with the actual data stored in the block. Instead of skipping the entire fractured block (see above) we can only skip the out-of-sync row.

3.11 Various parameters

3.11.1 VERSION

default value = "-1"
type = manditory
example : VERSION="92"

- defines the Oracle version the datafiles belong to
- valid values : 7, 80, 81, 91, 92, 101, 102, 111, 112

3.11.2 PLATFORM

default value = <not defined>
type = mandatory
example : PLATFORM="AIX"

- defines the platform the database runs/ran on
- cross-platform unloading is supported – for example if you FTP all datafiles from an IBM AIX server to a Windows box and run DUDE on the latter.
- should always be the platform the database originated from
- valid values : INTEL , AIX, SOLARIS, HPUX, TRU64 and VMS
- for Linux on Intel use "INTEL"
- for SunOS use "SOLARIS"
- for OpenVMS on alpha use "VMS"

3.11.3 PARTITIONING

default value = "false"
type = optional
example : PARTITIONING="true"

- if your database is installed with the partitioning option you should set this parameter to *true*
- DUDE supports partitions and subpartitions
- This parameter can not be set when *VERSION="7"*

3.11.4 ENABLE_LOBS

default value = "false"

type = optional
example : ENABLE_LOBS=”true”

- if your database contains tables with LOB columns (BLOB and/or CLOB) you should set this parameter to *true*
- make sure you specify a valid directory for *LOB_DIR* (cfr. *3.1.9 LOB_DIR*)

3.11.5 NO_FILESEP

default value = “false”
type = optional
example : NO_FILESEP=”true”

- This parameter should only be used where *System.getProperties().getProperty("file.separator")* returns an incorrect file separator like for example VMS

3.11.6 INCLUDE_BLOCK_DUMP

default value = “false”
type = optional
example : INCLUDE_BLOCK_DUMP=”true”

- In case an exception is thrown, and *INCLUDE_BLOCK_DUMP* is set to true, the logfile in *LOG_DIR* will include a hex blockdump
- Note that this will severely increase the amount of logging and will slow down processing.

3.12 Bootstrap parameters

The bootstrap parameters were introduced to support migrated datadictionaries generated by Oracle's "mig" utility. The *mig* is used to migrate an Oracle 7.x database to Oracle 8.0, 8i or 9i.
The *BOOTSTRAP* parameters set the object id's and table numbers for the base dictionary tables.

3.12.1 BOOTSTRAP_OBJ_OID

default value = "18"
type = optional
example : BOOTSTRAP_OBJ_OID="2407"

- Sets the data object id for base dictionary table OBJ$
- Set this parameter to *"17"* for Oracle 7
- This parameter should only be set when :
    - Extracting an Oracle 7 dictionary
    - Extracting a dictionary from a database that was migrated using Oracle's *mig* utility

3.12.2 BOOTSTRAP_COBJ_OID

default value = "2"
type = optional
example : BOOTSTRAP_COBJ_OID="2395"

- Sets the data object id for base dictionary clustered table C_OBJ$
- Set this parameter to *"1"* for Oracle 7
- This parameter should only be set when :
    - Extracting an Oracle 7 dictionary
    - Extracting a dictionary from a database that was migrated using Oracle's *mig* utility

3.12.3 BOOTSTRAP_CUSER_OID

default value = "10"
type = optional
example : BOOTSTRAP_CUSER_OID="2399"

- Sets the data object id for base dictionary clustered table C_USER$
- Set this parameter to *"9"* for Oracle 7
- This parameter should only be set when :
    - Extracting an Oracle 7 dictionary
    - Extracting a dictionary from a database that was migrated using Oracle's *mig* utility

3.12.4 BOOTSTRAP_TAB_TABNO

default value = "1"
type = optional
example : BOOTSTRAP_TAB_TABNO="1"

- Sets the table number for TAB$ within clustered table C_OBJ$
- Set this parameter to *"1"* for Oracle 7
- This parameter should only be set when :
    - Extracting an Oracle 7 dictionary
    - Extracting a dictionary from a database that was migrated using Oracle's *mig* utility

3.12.5 BOOTSTRAP_COL_TABNO

default value = "5"
type = optional
example : BOOTSTRAP_COL_TABNO="12"

- Sets the table number for COL$ within clustered table C_OBJ$
- Set this parameter to *"5"* for Oracle 7
- This parameter should only be set when :
    - Extracting an Oracle 7 dictionary
    - Extracting a dictionary from a database that was migrated using Oracle's *mig* utility

3.12.6 BOOTSTRAP_LOB_TABNO

default value = "8" (if VERSION="80"), "9" (if VERSION="81") and "6" (for all other VERSIONs)

type = optional
example : BOOTSTRAP_LOB_TABNO="12"

- Sets the table number for LOB$ within clustered table C_OBJ$
- This parameter should only be set when :
    - Extracting a dictionary from a database that was migrated using Oracle's *mig* utility

3.12.7 BOOTSTRAP_USER_TABNO

default value = "1"
type = optional
example : BOOTSTRAP_USER_TABNO="1"

- Sets the table number for COL$ within clustered table C_OBJ$
- Set this parameter to *"1"* for Oracle 7
- This parameter should only be set when :
    - Extracting an Oracle 7 dictionary
    - Extracting a dictionary from a database that was migrated using Oracle's *mig* utility

3.12.8 BOOTSTRAP_CFILEBLOCK_OID

default value = "1"
type = optional
example : BOOTSTRAP_CFILEBLOCK_OID="8"

- Sets the data object id for base dictionary clustered table C_FILE#_BLOCK#
- This parameter should only be set when :
    - Extracting a dictionary from a database that was migrated using Oracle's *mig* utility

3.12.9 BOOTSTRAP_SOURCE_OID

default value = "-1"
type = optional
example : BOOTSTRAP_SOURCE_OID="71"

- Sets the data object id for base dictionary table SOURCE$
- This parameter should only be set when :
    - Base dictionary tables are heavily corrupted, and the object id for SOURCE$ can not be retrieved via obj$

3.13 Deprecated parameters

3.13.1 BIG_ENDIAN

default value = depends on PLATFORM
type = optional
example : BIG_ENDIAN="true"

- this parameter should not be used anymore
- the endian parameter is automatically set when a *PLATFORM* is specified

3.14 Hidden parameters

Hidden parameters should not be used unless told so.

3.14.1 _SKIP_BLOB_TABLE

default value = "false"
type = optional
example : _SKIP_BLOB_TABLE="true"

- this parameter is legacy and should not be used
- was originally introduced to skip tables with BLOB columns

3.14.2 _FILENUMBER_BLOCK

default value = "0"
type = optional
example : _FILENUMBER_BLOCK="2"

- this parameter should only be used if the datafile header block is corrupt
- a corrupt datafile header block can lead to the incorrect resolving of the datafile number
- increase this parameter until you get a correct file number back during initialisation

3.15 Example dude.cfg

```
VERSION=”102”
// VERSION can be 7, 80, 81, 91, 92, 101, 102, 111, 112 (mandatory since DUDE 2.8.4)

// sizing parameters
BUFFERCACHE = "1000"
BLOCKSIZE = "8192"

// location parameters
SCAN_DIR = "c:\\oracle_stuff\\dude_scan"
LOG_DIR = "c:\\oracle_stuff\\dude_log"
DUMP_DIR = "c:\\oracle_stuff\\dude_dump"
PLSQL_DIR = "c:\\oracle_stuff\\dude_plsql"
SQLLDR_DIR = "c:\\oracle_stuff\\dude_ctl"
DDL_DIR = "c:\\oracle_stuff\\dude_ddl"
SCRIPT_DIR = "c:\\oracle_stuff\\dude_scan"
LOB_DIR = "c:\\oracle_stuff\\dude_lobs"
DICTIONARY_DIR = "c:\\oracle_stuff\\dude_dic"

// logfile name
LOG_FILE = "dude.log"

// cross-platform unloading
MAP_LOB_DIR="/lobs"
MAP_DUMP_DIR="/dump"
MAP_SQLLDR_DIR="/sqlldr"
MAP_FILESEP="/"

// replace dollar signs with underscore
PRETTY_FILE_NAMES="true"

// auto commands at startup
AUTO_FILENUMBER="true"
AUTO_DICTIONARY="true"
AUTO_BLOCK_CHECK="true"

// platform
PLATFORM="INTEL"  // INTEL/AIX/SOLARIS/HPUX/TRU64/VMS

PARTITIONING="true" // set to "true" if the database contains partitioned tables
ENABLE_LOBS="true" // set to "true" if the database contains LOB columns

EXPORT_MODE="false" // Export Mode (false = flatfile, true = Oracle 805 DMP file)
// the following parameters are evaluated when EXPORT_MODE="true"
EXPORT_FILE_SIZE="1048576000"
RECORDLENGTH="65536" /// "65536"
EXPORT_GZIP="false"
GZIP_BUFFER = "5242880"
EXPORT_NLS="WE8ISO8859P1"
EXPORT_NCHAR_NLS="UTF8"
// EOF EXPORT_MODE="true" parameters

// sqlloader flatfile specific
GEN_SQLLDR_CTL="true"
GEN_SQLLDR_STR_MODE="true"  // Generate recordset separator clause in SQLLDR controlfile
(this is only supported in sqlldr 8.1.6 and higher!)
                            // Set to "false" if loading into Oracle 8.0.x
RECORD_SEP="|||\n"   // RECORD SEPERATOR => IMPORTANT : if GEN_SQLLDR_STR_MODE=FALSE
then you must set to "\n" for UNIX and "\r\n" for WINDOWS
FIELD_SEP="44"
//FIELD_ENCL="34" // "255"
FIELD_ENCL="34"
NULL_FIELD=""
```

```
FLAT_NLS="ISO8859_1"

// generate table DDL scripts
GEN_TABLE_DDL="true"

TABLESPACE "SYSTEM"
{
        DATAFILE="C:\\ORACLE\\ORADATA\\KVMB\\SYSTEM01.DBF"
}

TABLESPACE "EXAMPLE"
{
        DATAFILE="C:\\ORACLE\\ORADATA\\KVMB\\EXAMPLE01.DBF"
        ASSM="true"
}


TABLESPACE "USERS"
{
        DATAFILE="C:\\ORACLE\\ORADATA\\KVMB\\USERS01.DBF"
        ASSM="true"
        SAMPLE_LIMIT="100" // this is only need if we don't have SYSTEM TS anymore
        SAMPLE_SIZE="1000" // this is only need if we don't have SYSTEM TS anymore
}
```

4 Getting started : DUDE

DUDE is delivered as one Java CLASS file and several DUDE files.
DUDE will be specifically compiled for you as it will contain your license information spread over several files. Therefore it is very important to inform me about the Java version you are running on your server. DUDE needs a JRE/JDK 1.4 environment to run.

4.1 Java version

To check the version of your java virtual machine run :
java -version
or for the java runtime environments
jre –version

IMPORTANT : make sure all DUDE (*.dude) files are writable ! You will not be able to start DUDE if they are not !
IMPORTANT : gnu java is not supported !

4.2 Running DUDE

java dude
or
jre dude

You might want to increase the maximum Java heap size when you start encountering out of memory exceptions (java.lang.OutOfMemoryError).
java –Xmx256m dude

You should see the following :

```
C:\dude_demo>java dude
License information :
User : Kurt Van Meerbeeck
Email : dude@ora600.org
Company : ORA600
Phone : +32-495-xxxxxxx
Fax : +32-15-xxxxxxx
Host : KOERT (192.168.123.3)
Platform : Windows XP x86
Valid for 7 days.


Current size logfile = 396370 bytes

jDUL/DUDE : 2.5.0

NOT FOR DISTRIBUTION 2001 - 2007 Kurt Van Meerbeeck - ORA600 - dude@ora600.org
```

```
FOR EDUCATIONAL USE ONLY !
PERSONAL COPY of Kurt Van Meerbeeck / ORA600 / dude@ora600.org

Roads ? Where we're going we don't need roads ...

DUDE> Initialising ...
DUDE> Init : creating filenumber map ...
DUDE> Scanning tablespace USERS : BLOCKSIZE = 8192
DUDE> File : C:\ORACLE\ORADATA\KVMB\USERS01.DBF resolves to number : 6
DUDE> Scanning tablespace EXAMPLE : BLOCKSIZE = 8192
DUDE> File : C:\ORACLE\ORADATA\KVMB\EXAMPLE01.DBF resolves to number : 3
DUDE> Scanning tablespace SYSTEM : BLOCKSIZE = 8192
DUDE> File : C:\ORACLE\ORADATA\KVMB\SYSTEM01.DBF resolves to number : 1
DUDE> Init : creating dictionary ...
DUDE> Error opening file : c:\oracle_stuff\dude_dic\user.dat
DUDE> Error opening file : c:\oracle_stuff\dude_dic\obj.dat
DUDE> Error opening file : c:\oracle_stuff\dude_dic\col.dat
DUDE> Error opening file : c:\oracle_stuff\dude_dic\tab.dat
DUDE> Error opening file : c:\oracle_stuff\dude_dic\tabpart.dat
DUDE> Error opening file : c:\oracle_stuff\dude_dic\tabsubpart.dat
DUDE> Error opening file : c:\oracle_stuff\dude_dic\lob.dat
DUDE> Init : checking datafile blocksizes ...
DUDE> DATAFILE = C:\ORACLE\ORADATA\KVMB\USERS01.DBF OK : BLOCKSIZE = 8192 equals
 internal blocksize !
DUDE> DATAFILE = C:\ORACLE\ORADATA\KVMB\EXAMPLE01.DBF OK : BLOCKSIZE = 8192 equa
ls internal blocksize !
DUDE> DATAFILE = C:\ORACLE\ORADATA\KVMB\SYSTEM01.DBF OK : BLOCKSIZE = 8192 equal
s internal blocksize !
DUDE> Init done.
DUDE>
```

Note that :

- I've set *AUTO_FILENUMBER="true"* in *dude.cfg* – thus during initializing the file# is resolved
- I've set *AUTO_DICTIONARY="true"* in *dude.cfg* – thus error messages are to be expected when no dictionary exists

4.3 Statements

All statements/commands in DUDE are :

- semi-colon (;) delimited !
- are **not case-sensitive**, **except** for **tablespace**, **user** and **table** identifiers !

5 Blockmaps

DUDE may or may not depend on extent maps stored in the datafiles. DUDE can use the **extent map** (*see also chapter 6 Extent maps)* information stored in segment header and extent map blocks or it is capable of building its own segment/extent information by creating **blockmaps**. In short, a blockmap is a list of blocks for a specific segment.

There are exceptions :

*If the **SYSTEM** tablespace **exists,** a **blockmap** for **tablespace SYSTEM must *always*** be **available** to bootstrap the dictionary within DUDE*

*If the **SYSTEM** tablespace **does not exist,** a **blockmap for all remaining tablespaces must *always*** be **available** in order to sample the data*

When a datafile is corrupted – there is a small chance that some segment headers or extent map blocks are corrupted too. The segment header blocks and extent map blocks contains extent information covering multiple blocks related to the segment. Corruption in segment header and extent map blocks therefore can have a large impact on the recoverability of the whole extent.

It is recommended to use **blockmap** in favor of **extent maps** in case of severe corruption.

A blockmap is a list of blocks sorted per OBJECTID.
Before doing any kind of operation on a tablespace, a blockmap should exist for that tablespace !

A **blockmap** only has to be created **once** for **each tablespace**.

In conclusion:

- Always create a blockmap for SYSTEM
- Always create a blockmap for all tablespaces
  - if you've lost SYSTEM
  - are uncertain about the severity of corruption of segment headers and extent map blocks
- set *USE_SEGMENT_HEADER="false"* if you use blockmaps

5.1 CREATE BLOCKMAP FOR TABLESPACE statement

This statement will create a block for the specified tablespace :
Example :

```
DUDE> create blockmap for tablespace SYSTEM ;
DUDE> ID := 0 BLOCKMAPPER for TABLESPACE NAME = SYSTEM
OFFSET = 0
ASSM = false
BIGFILE = false
BLOCKSIZE = 8192
NUMBER = 1 FILENAME = C:\ORACLE\ORADATA\KVMB\SYSTEM01.DBF IBS = 8192


DUDE> Datafile C:\ORACLE\ORADATA\KVMB\SYSTEM01.DBF : start reading blocks ...
DUDE> Datafile C:\ORACLE\ORADATA\KVMB\SYSTEM01.DBF : reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Please wait for chunk stream to finish ...
DUDE> Done !


DUDE> create blockmap for tablespace EXAMPLE ;
DUDE> ID := 0 BLOCKMAPPER for TABLESPACE NAME = EXAMPLE
OFFSET = 0
ASSM = true
BIGFILE = false
BLOCKSIZE = 8192
NUMBER = 3 FILENAME = C:\ORACLE\ORADATA\KVMB\EXAMPLE01.DBF IBS = 8192


DUDE> Datafile C:\ORACLE\ORADATA\KVMB\EXAMPLE01.DBF : start reading blocks ...
DUDE> Datafile C:\ORACLE\ORADATA\KVMB\EXAMPLE01.DBF : reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Please wait for chunk stream to finish ...
DUDE> Done !
```

After creating a blockmap for tablespace SYSTEM you can :

- dump the dictionary
- dump other data from SYSTEM
- create a dictionary
- dump plsql code

After creating a blockmap for tablespace EXAMPLE you can :

- dump data from EXAMPLE

If you take the time to look at the DICTIONARY_DIR you'll see all the blockmaps.

As of DUDE version 2.5.5, a summary will be shown after each datafile scanned. The summary shows the number of blocks/kilobytes scanned per blocktype.
This feature was introduced after having encountered several cases with zero'd out datablocks.
Imagine that you have 4Gb datafile – and you know that it contains at least 3Gb worth of data. If you see that the blockscanner only detects, let's say 500Mb, then you know immediately somethings wrong.

```
DUDE> create blockmap for tablespace USERS ;
DUDE> ID := 0 BLOCKMAPPER for TABLESPACE NAME = USERS
OFFSET = 0
```

```
ASSM = true
BIGFILE = false
BLOCKSIZE = 8192
NUMBER = 6 FILENAME = C:\ORACLE\ORADATA\KVMB\users01.dbf IBS = 8192


DUDE> Datafile C:\ORACLE\ORADATA\KVMB\users01.dbf : start reading blocks ...
DUDE> Datafile C:\ORACLE\ORADATA\KVMB\users01.dbf : reading blocks done !
DUDE> Block type 0 : 39872 blocks -> 318976Kb -> 39%
DUDE> Block type 6 : 45831 blocks -> 366648Kb -> 45%
DUDE> Block type 11 : 1 blocks -> 8Kb -> 0%
DUDE> Block type 29 : 1 blocks -> 8Kb -> 0%
DUDE> Block type 30 : 6 blocks -> 48Kb -> 0%
DUDE> Block type 32 : 6021 blocks -> 48168Kb -> 6%
DUDE> Block type 33 : 5332 blocks -> 42656Kb -> 5%
DUDE> Block type 35 : 5332 blocks -> 42656Kb -> 5%
DUDE> Block type 40 : 5 blocks -> 40Kb -> 0%
DUDE> Total Blocks scanned = 102401
DUDE> Please wait for output stream to finish ...
DUDE> Please wait for chunk stream to finish ...
DUDE> Done !
```

5.2 DROP BLOCKMAP FOR TABLESPACE statement

If you want to recreate a blockmap, you'll first have to drop the existing blockmap :

```
DUDE> create blockmap for tablespace SYSTEM ;
DUDE> illegal command or option
DUDE> blockmap for tablespace SYSTEM already exists ! (drop blockmap first)
DUDE> drop blockmap for tablespace SYSTEM ;
DUDE>
```

6 Extent maps

As explained in the previous chapter, DUDE can either use its own blockmaps to dump data, or it can use Oracle's extent map information stored in the segment's header block and extent map blocks.

In this chapter we'll have a closer look at using Oracle's extent maps.

6.1 Pro and contra

Why would you want to use Oracle extent maps ?
Short answer – because unloading data is faster.

Why shouldn't you use Oracle extent maps ?
Short answer – because it's less reliant.
Long answer – extent maps are pointers to extents – these extents contain datablocks. If the pointers to the data are corrupted, you will not see that data. This is true for all data unloaders on the market.

So – if you are facing massive corruption due to a raid crash, or file system corruption where parts of the datablocks are overwritten, zero'd out or contain plain garbage – I would strongly suggest not using extent maps !

If *FLAG_FRACTURED_BLOCK="true"*, then DUDE will generate a warning if something is wrong with the integrity of the block. It will do this by calculating a checksum and comparing this with the checksum stored within the block. This checksum is also calculated for all segment headers and extent/pagetable blocks. If you see these messages appear – stop immediately and switch completely to blockmaps.

6.2 SYSTEM tablespace considerations

As mentioned before, but repeated here :

*If the **SYSTEM** tablespace **exists,** a **blockmap** for **tablespace SYSTEM must *always*** be **available** to bootstrap the dictionary within DUDE*

*If the **SYSTEM** tablespace **does not exist,** a **blockmap for all remaining tablespaces must *always*** be **available** in order to sample the data*

6.3 Extentmap configuration

7 Do you speak Oracle – the dictionary

Now that you've created a blockmap for tablespace SYSTEM, you can start extracting the Oracle data dictionary by using the *DUMP DICTIONARY* command :

7.1 DUMP DICTIONARY

```
DUDE> dump dictionary ;
DUDE> Dumping OBJ$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows    = 25351
Migrated rows  = 0
Chained rows   = 0
Deleted rows    = 56 (not extracted!)
Skipped rows   = 0 (not extracted!)

DUDE> Done !
DUDE> Dumping TAB$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Done !
DUDE> Dumping TABPART$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows    = 55
Migrated rows  = 0
Chained rows   = 0
Deleted rows    = 23 (not extracted!)
Skipped rows   = 0 (not extracted!)

DUDE> Done !
DUDE> Dumping TABSUBPART$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows    = 24
Migrated rows  = 0
Chained rows   = 0
Deleted rows    = 0 (not extracted!)
Skipped rows   = 0 (not extracted!)

DUDE> Done !
DUDE> Dumping INDPART$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows    = 56
Migrated rows  = 0
Chained rows   = 0
Deleted rows    = 80 (not extracted!)
Skipped rows   = 0 (not extracted!)

DUDE> Done !
DUDE> Dumping IND$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
```

```
DUDE> Done !
DUDE> Dumping COL$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Done !
DUDE> Dumping USER$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Done !
DUDE> Dumping LOB$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Done !
```

If you take a look at the DICTIONARY_DIR you should see a couple of *.dat files :

- obj.dat
- tab.dat
- col.dat
- user.dat
- tabpart.dat (optional – when *PARTITIONING="true"*)
- tabsubpart.dat (optional – when *PARTITIONING="true"*)
- lob.dat (optional – when *LOBS_ENABLED="true"*)
- other *.dat are there for future releases

7.2 CREATE DICTIONARY

Once this is done, you can continue creating the DUDE dictionary. During this process the *.dat files created earlier are parsed into DUDE. To create the dictionary, use the *CREATE DICTIONARY* command :

```
DUDE> create dictionary ;
DUDE>
```

Now you can switch the parameter *AUTO_DICTIONARY* from *"false"* to *"true"* (if it wasn't already). Doing this, the *CREATE DICTIONARY* command is run automatically each time DUDE is started.

So, now you have created a dictionary, you should also be able to describe tables by using the *SHOW* commands. (see 8 Getting information out of your dictionary)

7.3 CROSSCHECK DICTIONARY

In some cases, the base dictionary can become corrupted. This may result in errors loading dictionary data into DUDE using the *CREATE DICTIONARY* command – and

even worse losing importing meta data. In these rare cases, it might be necessary to edit the *.dat files found in *DICTIONARY_DIR*.
The command *CROSSCHECK DICTIONARY* helps you with this, as it will crosscheck relations between obj$ and tab$, and give you instruction to edit the necessary *.dat files.

```
Example :
DUDE> crosscheck dictionary ;
DUDE> Checking tables in obj$ not found in tab$ ...
DUDE> Checking tables in obj$ not found in tab$ done !
DUDE> Add the following lines to tab.dat in C:\dude_dic
DUDE> All clear ! No lines to add !
DUDE>
```

8 Getting information out of your dictionary

Once the dictionary is created, you'll be able to query the dictionary by using the *SHOW* command.

8.1 SHOW OBJECTID <objectid>

The *SHOW OBJECTID* command will give you more information about the table identified by objectid <*objectid*>.
The dictionary must be available for this command to run.

```
Example - IOT
DUDE> show objectid 54576 ;
DUDE> OWNER = IT NAME = IOTNOOVERFLOW
 OBJECTID = 54576 DATA OBJECTID = -1 IOT OBJECTID = 54577
COL 1 [1] OBJECTID NUMBER
COL 2 [2] OWNER VARCHAR2 [WE8MSWIN1252]
COL 3 [3] OBJECT_NAME VARCHAR2 [WE8MSWIN1252]
Table Properties :
->index-only table (IOT)

Example - partitioned heap table
DUDE> show objectid 25577 ;
DUDE> OWNER = SH NAME = SALES
 OBJECTID = 25577 DATA OBJECTID = -1  *PARTITIONED*
COL 1 [1] PROD_ID NUMBER
COL 2 [2] CUST_ID NUMBER
COL 3 [3] TIME_ID DATE
COL 4 [4] CHANNEL_ID CHAR [WE8MSWIN1252]
COL 5 [5] PROMO_ID NUMBER
COL 6 [6] QUANTITY_SOLD NUMBER
COL 7 [7] AMOUNT_SOLD NUMBER
NAME = SALES_1995 OWNER = SH TABLE = SALES OBJ# = 25578 DATAOBJ# = 25578 BO# = 25577
NAME = SALES_1996 OWNER = SH TABLE = SALES OBJ# = 25579 DATAOBJ# = 25579 BO# = 25577
NAME = SALES_H1_1997 OWNER = SH TABLE = SALES OBJ# = 25580 DATAOBJ# = 25580 BO# = 25577
NAME = SALES_H2_1997 OWNER = SH TABLE = SALES OBJ# = 25581 DATAOBJ# = 25581 BO# = 25577
NAME = SALES_Q1_1998 OWNER = SH TABLE = SALES OBJ# = 25582 DATAOBJ# = 25582 BO# = 25577
NAME = SALES_Q2_1998 OWNER = SH TABLE = SALES OBJ# = 25583 DATAOBJ# = 25583 BO# = 25577
NAME = SALES_Q3_1998 OWNER = SH TABLE = SALES OBJ# = 25584 DATAOBJ# = 25584 BO# = 25577
NAME = SALES_Q4_1998 OWNER = SH TABLE = SALES OBJ# = 25585 DATAOBJ# = 25585 BO# = 25577
NAME = SALES_Q1_1999 OWNER = SH TABLE = SALES OBJ# = 25586 DATAOBJ# = 25586 BO# = 25577
NAME = SALES_Q2_1999 OWNER = SH TABLE = SALES OBJ# = 25587 DATAOBJ# = 25587 BO# = 25577
NAME = SALES_Q3_1999 OWNER = SH TABLE = SALES OBJ# = 25588 DATAOBJ# = 25588 BO# = 25577
NAME = SALES_Q4_1999 OWNER = SH TABLE = SALES OBJ# = 25589 DATAOBJ# = 25589 BO# = 25577
NAME = SALES_Q1_2000 OWNER = SH TABLE = SALES OBJ# = 25590 DATAOBJ# = 25590 BO# = 25577
NAME = SALES_Q2_2000 OWNER = SH TABLE = SALES OBJ# = 25591 DATAOBJ# = 25591 BO# = 25577
NAME = SALES_Q3_2000 OWNER = SH TABLE = SALES OBJ# = 25592 DATAOBJ# = 25592 BO# = 25577
NAME = SALES_Q4_2000 OWNER = SH TABLE = SALES OBJ# = 25593 DATAOBJ# = 25593 BO# = 25577
Table Properties :
->partitioned table
->pdml itl invariant


Example - partitioned IOT with overflow segment
DUDE> show objectid 54581 ;
DUDE> OWNER = IT NAME = IOTPAR
```

```
 OBJECTID = 54581 DATA OBJECTID = -1 BASE OBJECTID = 54586 IOT OBJECTID = NULL
*PARTITIONED*
COL 1 [1] OBJECTID NUMBER
COL 2 [2] OWNER VARCHAR2 [WE8MSWIN1252]
COL 3 [3] OBJECT_NAME VARCHAR2 [WE8MSWIN1252]
NAME = IOT1 OWNER = IT TABLE = IOTPAR OBJ# = 54582 DATAOBJ# = -1 BO# = 54581
NAME = IOT2 OWNER = IT TABLE = IOTPAR OBJ# = 54583 DATAOBJ# = -1 BO# = 54581
NAME = IOT3 OWNER = IT TABLE = IOTPAR OBJ# = 54584 DATAOBJ# = -1 BO# = 54581
NAME = IOT4 OWNER = IT TABLE = IOTPAR OBJ# = 54585 DATAOBJ# = -1 BO# = 54581
Table Properties :
->partitioned table
->index-only table (IOT)
->IOT w/ row overflow
```

8.2 SHOW TABLE <username> <tablename>

The *SHOW TABLE* command will give you more information about the table <*tablename*>, owned by user <*user*>.
The dictionary must be available for this command to run.

```
Example - normal heap table
DUDE> show table SH PROMOTIONS ;
DUDE> OWNER = SH NAME = PROMOTIONS
 OBJECTID = 25569 DATA OBJECTID = 25748
COL 1 [1] PROMO_ID NUMBER
COL 2 [2] PROMO_NAME VARCHAR2 [WE8MSWIN1252]
COL 3 [3] PROMO_SUBCATEGORY VARCHAR2 [WE8MSWIN1252]
COL 4 [4] PROMO_CATEGORY VARCHAR2 [WE8MSWIN1252]
COL 5 [5] PROMO_COST NUMBER
COL 6 [6] PROMO_BEGIN_DATE DATE
COL 7 [7] PROMO_END_DATE DATE
Table Properties :
->pdml itl invariant
```

As you can see – the *SHOW* command will basically describe the object if it is found in the dictionary, based on **objectid** or **object owner and name**.
The description consists of :

- objected
- dataobjectid
- object owner and name
- columns
- column logical and physical order
- characterset for CHAR, VARCHAR and CLOB columns
- partitions (if partitioned)
  - partition name
  - partition objectid
  - partition dataobjectid
  - partition base objectid
- table properties
  - type of table (normal/partitioned/iot)
  - if table contains lobs
  - if table contains user defined datatypes
  - has overflow segments (iot)

8.3 SHOW USERS

The *SHOW USERS* will return all users and roles found in the dictionary.
The dictionary must be available for this command to run.

8.4 SHOW TABLESPACES

The *SHOW TABLESPACES* will return all tablespaces defined in *dude.cfg*.

8.5 SHOW TABLESPACE <tablespace name>

The *SHOW TABLESPACE* command will return the parameters from tablespace <*tablespace name*> defined in *dude.cfg*.

```
DUDE> show tablespace USERS ;
DUDE> TABLESPACE NAME = USERS
OFFSET = 0
ASSM = true
BIGFILE = false
BLOCKSIZE = 8192
NUMBER = 4 FILENAME = C:\ORACLE\PRODUCT\10.2.0\ORADATA\KVMB\USERS01.DBF IBS = 8192
```

8.6 SHOW TABLESPACES FROM DICTIONARY

The previous command *SHOW TABLESPACE* only shows the tablespace parameters as defined in *dude.cfg*. The command *SHOW TABLESPACES FROM DICTIONARY* will show tablespace properties as defined in the Oracle dictionary. Before you can run this command a dictionary has to be available.

```
DUDE> show tablespaces from dictionary ;
DUDE> ------------------------------------------------------------------------
DUDE> Tablespace ts# = 6  -  name = EXAMPLE  -  blocksize = 8Kb
DUDE> Tablespace properties :
DUDE> ->system managed allocation
->auto segment space management (ASSM=true)
DUDE> Tablespace datafiles :
DUDE> file# = 5 - rfile# = 5 - size = 100.0Mb

DUDE> ------------------------------------------------------------------------
DUDE> Tablespace ts# = 5  -  name = UNDOTBS2  -  blocksize = 8Kb
DUDE> Tablespace properties :
DUDE> ->system managed allocation
->undo tablespace
->freelist segment space management (ASSM=false)

DUDE> ------------------------------------------------------------------------
DUDE> Tablespace ts# = 4  -  name = USERS  -  blocksize = 8Kb
DUDE> Tablespace properties :
DUDE> ->system managed allocation
->auto segment space management (ASSM=true)
```

```
DUDE> Tablespace datafiles :
DUDE> file# = 4 - rfile# = 4 - size = 5.0Mb

DUDE> ------------------------------------------------------------------------
DUDE> Tablespace ts# = 3  -  name = TEMP  -  blocksize = 8Kb
DUDE> Tablespace properties :
DUDE> ->uniform allocation
->freelist segment space management (ASSM=false)

DUDE> ------------------------------------------------------------------------
DUDE> Tablespace ts# = 2  -  name = SYSAUX  -  blocksize = 8Kb
DUDE> Tablespace properties :
DUDE> ->system managed allocation
->auto segment space management (ASSM=true)
DUDE> Tablespace datafiles :
DUDE> file# = 3 - rfile# = 3 - size = 120.0Mb

DUDE> ------------------------------------------------------------------------
DUDE> Tablespace ts# = 1  -  name = UNDOTBS1  -  blocksize = 8Kb
DUDE> Tablespace properties :
DUDE> ->system managed allocation
->undo tablespace
->freelist segment space management (ASSM=false)
DUDE> Tablespace datafiles :
DUDE> file# = 2 - rfile# = 2 - size = 25.0Mb

DUDE> ------------------------------------------------------------------------
DUDE> Tablespace ts# = 0  -  name = SYSTEM  -  blocksize = 8Kb
DUDE> Tablespace properties :
DUDE> ->system managed allocation
->freelist segment space management (ASSM=false)
DUDE> Tablespace datafiles :
DUDE> file# = 1 - rfile# = 1 - size = 300.0Mb
```

As seen from the example – this command will show you what kind of tablespace it is, the allocation type and more importantly if the tablespace uses automatic segment space management or not (ASSM="true" or "false"), which in turn can be used to adjust your *dude.cfg*.
The command output also shows you the absolute and relative datafile numbers and the tablespace blocksizes.

8.7 SHOW HEADER <objectid>

This command outputs segment header information, such as :

- Segment type
- Location of the header block (tablespace number/file number/block offset)
- Number of extents
- Number of blocks
- High watermark address
- Number of extent maps
- Extent addresses and size in blocks

```
DUDE> show header 51151 ;
DUDE> Retrieving header block ts# = 4 file# = 4 block# = 27
DUDE> TYPE=23
HEADER: seg/obj=51151,exts=1, blks=8
HWM   : address=01000021, ext#=0, blocks=8, ext size=8, flg blocks=0, flg bellow
=5
# maps = 0, # exts in this header = 1
Next Map : file#=0, offset=0
Extent:
0 01000019      8
```

9 Did you say Oracle 6/7 – the migrated dictionary

Once, a long long time ago, when animals could still speak, there were such things like Oracle version 6 and 7. Since then, the Oracle database has evolved – certain things have changed. For example -database block addressing, data dictionary tables, relative and absolute file numbers and so on.
When a database was migrated from Oracle 6/7 to 8.0, 8i or 9i, the dictionary had to be recreated. This was done using the *mig* utility in combination with the *migrate.bsq* script.
This meant that the (data)object id's, for the base dictionary tables, were completely different than when they were created from scratch with the *sql.bsq* script.

9.1 How do we know this was once an Oracle 7 beauty

In Oracle 6 there were only 5 to 6 bits used for the file number. So only a maximum of 2^5-1 (31) or 2^6-1 (63) datafiles could be used (database wide).
In Oracle 7 this changed to 10bits or 2^10-1 (1023) datafiles (database wide). However, because of backward compatibility with Oracle 6 an encoding scheme was introduced splitting up the 10bits for file number into 6 and 4 bits and wrapping them around. It really depends on the platform. On intel windows and IBM AIX for example, I've seen an 8/2 split.

```
SVRMGR> select dump(chartorowid('00000000.0000.0001')) from dual ;
DUMP(CHARTOROWID('0000000
-------------------------
Typ=69 Len=6: 1,0,0,0,0,0
1 row selected.
SVRMGR> select dump(chartorowid('00000000.0000.ffff')) from dual ;
DUMP(CHARTOROWID('00000000.00
-----------------------------
Typ=69 Len=6: 255,192,0,0,0,0
1 row selected.
```

This means that the first file# is :

```
00000001 00000000 00000000 00000000  -> file# 1
```

And the maximum file# is :

```
11111111 11000000 00000000 00000000  -> file# 1023
```

So the 10bits encoding scheme is like this :

```
LLLL LLLL HH
```

Where L is the low order bits
And H the high order bits

Now let's open DUDE on a series of these datafiles :

```
DUDE> Initialising ...
DUDE> Init : creating filenumber map ...
DUDE> Scanning tablespace SYSTEM : BLOCKSIZE = 2048
DUDE> File : G:\sys1orcl.ora resolves to number : 4
DUDE> File : G:\sys2.ora resolves to number : 40
```

You'll notice that *sys1orcl.ora* which is basically the first file of the database has file# equal to 4. And we know that sys2.ora had file# equal to 10. How's that possible ?

```
File# 1  = 0000 0001 00 (LLLL LLLL HH) EQUALS 4 in Oracle 8 DBA format
File# 10 = 0000 1010 00 (LLLL LLLL HH) EQUALS 40 in Oracle 8 DBA
format
```

It's clear that using the Oracle 8 DBA format encoding on the Oracle 7 wrapped DBA format, results in different file numbers. Basically, the file number shifted 2 bits to the left (or x2x2). This is of course platform specific, but if the first file of SYSTEM has a file number that is a multiple of 2, you probably have a migrated database.

So what happened when Oracle 8.0 came along and introduced 2^10-1 or 1023 datafiles *per tablespace* ! Well – the DBA format stayed the same. However, the file numbers became relative to the tablespace. So 2 datafiles of the same *database* could have potentially the same file number, but belong to 2 *different tablespaces*.
What happened to the Oracle 7 (absolute) file numbers when it was migrated to Oracle 8. Surely, the *mig* utility didn't update the DBA for all blocks ?
Let's check out an Oracle 7 database :

```
SVRMGR> desc file$
Column Name                    Null?    Type
------------------------------ -------- ----
FILE#                          NOT NULL NUMBER
STATUS$                        NOT NULL NUMBER
BLOCKS                         NOT NULL NUMBER
TS#                            NOT NULL NUMBER

SVRMGR> select file#, ts# from file$ ;
FILE#      TS#
---------- ----------
         1          0
         2          1
         3          2
         4          3
         5          8
         6          9
         7         10
         8          7
         9         11
        10          0
10 rows selected.
```

Ok – looks logical – we see that tablespace TS#=0 or SYSTEM has 2 datafiles with file#=1 and file#=10.

Let's do the same after a migration to 8.0 :

```
SVRMGR> desc file$
Column Name                    Null?    Type
------------------------------ -------- ----
FILE#                          NOT NULL NUMBER
STATUS$                        NOT NULL NUMBER
BLOCKS                         NOT NULL NUMBER
TS#                                     NUMBER
RELFILE#                                NUMBER
MAXEXTEND                               NUMBER
INC                                     NUMBER
CRSCNWRP                                NUMBER
CRSCNBAS                                NUMBER
OWNERINSTANCE                           VARCHAR2(30)
SPARE1                                  NUMBER
SPARE2                                  NUMBER
SPARE3                                  VARCHAR2(1000)
SPARE4                                  DATE

SVRMGR> select file#,ts# from file$ ;
FILE#      TS#
---------- ----------
         1          0
         2          1
         3          2
         4          3
         5          8
         6          9
         7         10
         8          7
         9         11
        10          0
10 rows selected.
```

So – the file# for the datafiles stayed the same. But we can see an add column in file$ - relfile# :

```
SVRMGR> select file#,relfile#,ts# from file$ ;
FILE#      RELFILE#   TS#
---------- ---------- ----------
         1          4          0
         2          8          1
         3         12          2
         4         16          3
         5         20          8
         6         24          9
         7         28         10
         8         32          7
         9         36         11
        10         40          0
10 rows selected.
```

Here we can clearly see the 2bit shift to the left – the Oracle 7 absolute filenumber became an Oracle 8 relative filenumber.
So the *mig* utility did **not** have to :

- update the DBA in a block
- row addresses in chained and migrated rows

9.2 SCAN BOOTSTRAP

Once we know that are database was once migrated, we have to get our hands dirty and go out on a quest for the base dictionary table's object id's.
The quest starts with finding *BOOTSTRAP$*.
BOOTSTRAP$* contains all the information we need. To find *BOOTSTRAP$ you need to issue the *SCAN BOOTSTRAP* command **after you've created a blockmap for tablespace SYSTEM**.
Also – make sure you have set *SAMPLE_LIMIT* and *SAMPLE_SIZE* within the tablespace clause of SYSTEM.

```
TABLESPACE "SYSTEM"
{

      DATAFILE="G:\\sys1orcl.ora"
      DATAFILE="G:\\sys2.ora"
      SAMPLE_LIMIT="1000"
      SAMPLE_SIZE="100"
}
```

```
DUDE> create blockmap for tablespace SYSTEM ;
DUDE> ID := 0 BLOCKMAPPER for TABLESPACE NAME = SYSTEM
OFFSET = 0
ASSM = false
BIGFILE = false
BLOCKSIZE = 2048
NUMBER = 4 FILENAME = G:\sys1orcl.ora IBS = 2048
NUMBER = 40 FILENAME = G:\sys2.ora IBS = 2048
DUDE> Datafile G:\chili\migrated734\A\sys1orcl.ora : start reading blocks ...
DUDE> Datafile G:\chili\migrated734\A\sys1orcl.ora : reading blocks done !
DUDE> Datafile G:\chili\migrated734\A\sys2.ora : start reading blocks ...
DUDE> Datafile G:\chili\migrated734\A\sys2.ora : reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Done !
```

```
DUDE> scan BOOTSTRAP ;

DUDE> Scanning OBJECTID : 2436
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE> Possbile BOOTSTRAP$ found = 2436
DUDE>
Rows sampled : 52
COL 0 : LEN = 3 : NULLS = 0% : POSS. DATE = 1% : PRINTABLE = 22% : NICE NUM = 10
0%
COL 1 : LEN = 3 : NULLS = 0% : POSS. DATE = 1% : PRINTABLE = 11% : NICE NUM = 10
0%
COL 2 : LEN = 1247 : NULLS = 0% : POSS. DATE = 1% : PRINTABLE = 99% : NICE NUM =
 1%

DUDE> Done !
DUDE> Scanning OBJECTID : 83
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE> Done !
```

```
DUDE> dump SCANNED OBJECTID 179 (  COL0 NUMBER ,  COL1 NUMBER ,  COL2 CHAR ) ;
DUDE> Dumping objectid 179
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows    = 0
Migrated rows  = 0
Chained rows   = 0
Deleted rows    = 0 (not extracted!)
Skipped rows   = 0 (not extracted!)

DUDE> Done !
DUDE> Dumping objectid 179 Done !
DUDE> dump SCANNED OBJECTID 2600 (  COL0 NUMBER ,  COL1 NUMBER ,  COL2 CHAR ) ;
DUDE> Dumping objectid 2600
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows    = 291767
Migrated rows  = 0
Chained rows   = 0
Deleted rows    = 45930 (not extracted!)
Skipped rows   = 0 (not extracted!)

DUDE> Done !
DUDE> Dumping objectid 2600 Done !
DUDE> dump SCANNED OBJECTID 2600 (  COL0 NUMBER ,  COL1 NUMBER ,  COL2 CHAR ) ;
DUDE> Dumping objectid 2600
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows    = 291767
Migrated rows  = 0
Chained rows   = 0
Deleted rows    = 45930 (not extracted!)
Skipped rows   = 0 (not extracted!)

DUDE> Done !
DUDE> Dumping objectid 2600 Done !
```

DUDE has now found a couple of objects that **could be bootstrap$**.
You'll have to check the data from these candidates in *DUMP_DIR* :

```
Directory of C:\oracle_stuff\dude_dump

12/24/2006  04:15 PM                 0 SCANNED_179.dat
12/24/2006  04:15 PM            19,196 SCANNED_2436.dat
12/24/2006  04:15 PM           181,454 SCANNED_2466.dat
12/24/2006  04:14 PM               499 SCANNED_2558.dat
12/24/2006  04:15 PM        14,479,708 SCANNED_2600.dat
12/24/2006  04:15 PM            11,405 SCANNED_43.dat
12/24/2006  04:15 PM        10,547,967 SCANNED_51.dat
12/24/2006  04:15 PM            43,257 SCANNED_71.dat
               8 File(s)     25,283,486 bytes
               0 Dir(s)   5,626,392,576 bytes free
```

Bootstrap$, typically has around 40 to 60 records in it. They are all create statements for dictionary objects.
In our case, when we opened the file *SCANNED_2436.dat* we clearly identified this file as the content of bootstrap$.

```
"13","2403","CREATE TABLE SEG$     (FILE#   NUMBER NOT NULL,      BLOCK#  NUMBER NOT NULL,      TYPE#    NUMBER NOT NULL,      TS#
NUMBER NOT NULL,      BLOCKS  NUMBER NOT NULL,      EXTENTS NUMBER NOT NULL,      INIEXTS NUMBER NOT NULL,      MINEXTS NUMBER NOT
NULL,      MAXEXTS NUMBER NOT NULL,      EXTSIZE NUMBER NOT NULL,      EXTPCT  NUMBER NOT NULL,      USER#   NUMBER NOT NULL,
LISTS   NUMBER,      GROUPS  NUMBER,      BITMAPRANGES NUMBER NOT NULL,      CACHEHINT NUMBER NOT NULL,      SCANHINT  NUMBER NOT
NULL,      HWMINCR   NUMBER NOT NULL,      SPARE1    NUMBER,      SPARE2    NUMBER)storage ( objno 2403 tabno 2) cluster
C_FILE#_BLOCK#(TS#, FILE#, BLOCK#)"|||

…

"16","2406","CREATE TABLE FILE$     (FILE#   NUMBER NOT NULL,      STATUS$ NUMBER NOT NULL,      BLOCKS  NUMBER NOT NULL,      TS#
NUMBER,      RELFILE# NUMBER,      MAXEXTEND NUMBER,      INC       NUMBER,      CRSCNWRP  NUMBER,      CRSCNBAS  NUMBER,
OWNERINSTANCE  VARCHAR2(30),      SPARE1          NUMBER,      SPARE2          NUMBER,      SPARE3          VARCHAR2(1000),      SPARE4
DATE) pctfree 10 pctused 40 initrans 1 maxtrans 255 storage ( initial 10240 next 10240 minextents 1 maxextents 121 pctincrease 50
objno 2406 extents ( file 40 block 242))"|||
"17","2407","CREATE TABLE OBJ$     ( OBJ#          NUMBER NOT NULL,       DATAOBJ#      NUMBER,       OWNER#        NUMBER NOT
NULL,       NAME          VARCHAR2(30) NOT NULL,       NAMESPACE     NUMBER NOT NULL,       SUBNAME       VARCHAR2(30),       TYPE#
NUMBER NOT NULL,       CTIME         DATE NOT NULL,       MTIME         DATE NOT NULL,       STIME         DATE NOT NULL,       STATUS
NUMBER NOT NULL,       REMOTEOWNER   VARCHAR2(30),       LINKNAME      VARCHAR2(128),       FLAGS         NUMBER,       OID$
RAW(16),       SPARE1        NUMBER,       SPARE2        NUMBER,       SPARE3        NUMBER,       SPARE4        VARCHAR2(1000),
SPARE5        VARCHAR2(1000),       SPARE6        DATE) pctfree 10 pctused 40 initrans 1 maxtrans 255 storage ( initial 10240 next
126976 minextents 1 maxextents 121 pctincrease 50 objno 2407 extents ( file 40 block 247))"|||

…

"19","2409","CREATE TABLE ICOL$     (OBJ#         NUMBER NOT NULL,      BO#          NUMBER NOT NULL,      COL#         NUMBER NOT
NULL,      POS#          NUMBER NOT NULL,      SEGCOL#      NUMBER NOT NULL,      SEGCOLLENGTH NUMBER NOT NULL,      OFFSET
NUMBER NOT NULL,      INTCOL#      NUMBER NOT NULL,      SPARE1       NUMBER,      SPARE2       NUMBER,      SPARE3       NUMBER,
SPARE4       VARCHAR2(1000),      SPARE5       VARCHAR2(1000),      SPARE6       DATE)storage ( objno 2409 tabno 4) cluster
C_OBJ#(BO#)"|||
"20","2607","CREATE TABLE COL$     (OBJ#          NUMBER NOT NULL,      COL#          NUMBER NOT NULL,      SEGCOL#       NUMBER
NOT NULL,      SEGCOLLENGTH  NUMBER NOT NULL,      OFFSET        NUMBER NOT NULL,      NAME          VARCHAR2 (30) NOT NULL,
TYPE#         NUMBER NOT NULL,      LENGTH        NUMBER NOT NULL,      FIXEDSTORAGE  NUMBER NOT NULL,      PRECISION#    NUMBER,
SCALE         NUMBER,      NULL$         NUMBER NOT NULL,      DEFLENGTH     NUMBER,      DEFAULT$      LONG,      INTCOL#
NUMBER NOT NULL,      PROPERTY      NUMBER NOT NULL,      CHARSETID     NUMBER,      CHARSETFORM   NUMBER,      SPARE1        NUMBER,
SPARE2        NUMBER,      SPARE3        NUMBER,      SPARE4        VARCHAR2(1000),      SPARE5        VARCHAR2(1000),      SPARE6
DATE)storage ( objno 2607 tabno 12) cluster C_OBJ#(OBJ#)"|||
"21","2410","CREATE TABLE USER$     (USER#     NUMBER NOT NULL,      NAME      VARCHAR2 (30) NOT NULL,      TYPE#      NUMBER NOT
NULL,      PASSWORD  VARCHAR2 (30),      DATATS#   NUMBER NOT NULL,      TEMPTS#   NUMBER NOT NULL,      CTIME     DATE NOT NULL,
PTIME     DATE,      EXPTIME   DATE,      LTIME     DATE,      RESOURCE$ NUMBER NOT NULL,      AUDIT$    VARCHAR2 (38),      DEFROLE
NUMBER NOT NULL,      DEFGRP#   NUMBER,      DEFGRP_SEQ# NUMBER,      ASTATUS   NUMBER NOT NULL,      LCOUNT    NUMBER NOT NULL,
DEFSCHCLASS VARCHAR2(30),      EXT_USERNAME VARCHAR2(4000),      SPARE1    NUMBER,      SPARE2    NUMBER,      SPARE3    NUMBER,
```

```
SPARE4    VARCHAR2(1000),      SPARE5    VARCHAR2(1000),      SPARE6    DATE)storage ( objno 2410 tabno 1) cluster
C_USER#(USER#)"|||
```

…

```
"30","2419","create unique index i_obj1 on obj$(obj#) pctfree 10 initrans 2 maxtrans 255 storage ( initial 10240 next 55296 minextents
1 maxextents 121 pctincrease 50 objno 2419 extents ( file 40 block 282))"|||
"1","2395","create cluster c_obj# (obj# number) pctfree 5 pctused 40 initrans 2 maxtrans 255 storage (
initial 10240 next 430080 minextents 1 maxextents 121 pctincrease 50 objno 2395 extents ( file 4 block
1305)) size 800"|||
"2","2396","create index i_obj# on cluster c_obj# pctfree 10 initrans 2 maxtrans 255 storage ( initial 10240 next 16384 minextents 1
maxextents 121 pctincrease 50 objno 2396 extents ( file 40 block 202))"|||
"3","2397","CREATE TABLE TAB$      (OBJ#          NUMBER NOT NULL,       DATAOBJ#      NUMBER,       TS#           NUMBER NOT NULL,
FILE#         NUMBER NOT NULL,       BLOCK#        NUMBER NOT NULL,       BOBJ#         NUMBER,       TAB#          NUMBER,       COLS
NUMBER NOT NULL,       CLUCOLS       NUMBER,       PCTFREE$      NUMBER NOT NULL,       PCTUSED$      NUMBER NOT NULL,       INITRANS
NUMBER NOT NULL,       MAXTRANS      NUMBER NOT NULL,       FLAGS         NUMBER NOT NULL,       AUDIT$        VARCHAR2(38) NOT NULL,
ROWCNT        NUMBER,       BLKCNT        NUMBER,       EMPCNT        NUMBER,       AVGSPC        NUMBER,       CHNCNT        NUMBER,
AVGRLN        NUMBER,       AVGSPC_FLB    NUMBER,       FLBCNT        NUMBER,       ANALYZETIME   DATE,       SAMPLESIZE    NUMBER,
DEGREE        NUMBER,       INSTANCES     NUMBER,       INTCOLS       NUMBER NOT NULL,       KERNELCOLS    NUMBER NOT NULL,
PROPERTY      NUMBER NOT NULL,       TRIGFLAG      NUMBER,       SPARE1        NUMBER,       SPARE2        NUMBER,       SPARE3
NUMBER,       SPARE4        VARCHAR2(1000),       SPARE5        VARCHAR2(1000),       SPARE6        DATE)storage ( objno 2397 tabno
1) cluster C_OBJ#(OBJ#)"|||
```

…

```
"8","2392","create index i_file#_block# on cluster c_file#_block# pctfree 10 initrans 2 maxtrans 255 storage ( initial 61440 next
24576 minextents 1 maxextents 121 pctincrease 50 objno 2392 extents ( file 40 block 142))"|||
"9","2399","create cluster c_user# (user# number) pctfree 10 pctused 40 initrans 2 maxtrans 255 storage ( initial 10240 next
16384 minextents 1 maxextents 121 pctincrease 50 objno 2399 extents ( file 40 block 207)) size 372"|||
"10","2400","create index i_user# on cluster c_user# pctfree 10 initrans 2 maxtrans 255 storage ( initial 10240 next 10240 minextents
1 maxextents 121 pctincrease 50 objno 2400 extents ( file 40 block 212))"|||
"11","2401","CREATE TABLE FET$      (TS#    NUMBER NOT NULL,       FILE#  NUMBER NOT NULL,       BLOCK# NUMBER NOT NULL,       LENGTH
NUMBER NOT NULL)storage ( objno 2401 tabno 1) cluster C_TS#(TS#)"|||
"12","2402","CREATE TABLE UET$     (SEGFILE#  NUMBER NOT NULL,       SEGBLOCK# NUMBER NOT NULL,       EXT#      NUMBER NOT NULL,
TS#       NUMBER NOT NULL,       FILE#     NUMBER NOT NULL,       BLOCK#    NUMBER NOT NULL,       LENGTH    NUMBER NOT NULL)storage (
objno 2402 tabno 1) cluster C_FILE#_BLOCK#(TS#, SEGFILE#, SEGBLOCK#)"|||
```

In this flatfile is first column represents *LINE#* - we're not interested in that. We need *OBJ#* (second column) and *SQL_TEXT* (third column).
More particular – we need to find out :

- object id for OBJ$ = 2407
- object id for C_OBJ# = 2395
- object id for C_USER# = 2399
- table number for TAB$ = 1
- table number for COL$ = 12
- table number for USER$ = 1

Knowing this – we can now add the following parameters to *dude.cfg* :

```
BOOTSTRAP_OBJ_OID       = "2407"
BOOTSTRAP_COBJ_OID      = "2395"
BOOTSTRAP_CUSER_OID     = "2399"
BOOTSTRAP_TAB_TABNO     = "1"
BOOTSTRAP_COL_TABNO     = "12"
BOOTSTRAP_USER_TABNO    = "1"
```

If you now restart DUDE and issue the *DUMP DICTIONARY* command, our dictionary will be correctly identified and unloaded :

```
DUDE> dump dictionary ;
DUDE> Dumping OBJ$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows    = 3261
Migrated rows  = 0
Chained rows   = 0
Deleted rows    = 16 (not extracted!)
Skipped rows   = 0 (not extracted!)

DUDE> Done !
DUDE> Dumping TAB$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Done !
DUDE> Dumping IND$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Done !
DUDE> Dumping COL$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Done !
DUDE> Dumping USER$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Done !
```

The dictionary is unload – so we can now created it using the *CREATE DICTIONARY* command :

```
DUDE> create dictionary ;
DUDE>
```

10 Dumping data

Data can be extracted using following commands :

10.1 DUMP OBJECTID

Table per table extraction can be done using the *DUMP OBJECTID* command :

```
DUDE> dump objectid 25569 ;
DUDE> Dumping OWNER = SH NAME = PROMOTIONS
 OBJECTID = 25569 DATA OBJECTID = 25748
COL 1 [1] PROMO_ID NUMBER
COL 2 [2] PROMO_NAME VARCHAR2 [WE8MSWIN1252]
COL 3 [3] PROMO_SUBCATEGORY VARCHAR2 [WE8MSWIN1252]
COL 4 [4] PROMO_CATEGORY VARCHAR2 [WE8MSWIN1252]
COL 5 [5] PROMO_COST NUMBER
COL 6 [6] PROMO_BEGIN_DATE DATE
COL 7 [7] PROMO_END_DATE DATE

DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows     = 501
Migrated rows   = 0
Chained rows    = 0
Deleted rows     = 0 (not extracted!)
Skipped rows    = 0 (not extracted!)

DUDE> Done !
```

The *DUMP* will give you feedback on :

- the number of normal rows extracted
- the number of migrated rows extracted
- the number of chained rows extracted
- the number of deleted rows that were **not extracted**
- the number of skipped rows that were **not extracted**

The *DUMP* command will generate a file called *SH_PROMOTIONS_25569.dat* in the *DUMP_DIR* directory if *EXPORT_MODE="false"* :

```
"1","promotion name# 1","downtown billboard","post","77200","15-09-1998 ad 00:00:00","15-11-1998 ad 00:00:00"|||
"2","promotion name# 2","hospital flyer","flyer","47100","13-04-1999 ad 00:00:00","13-07-1999 ad 00:00:00"|||
"3","promotion name# 3","coupon news","newspaper","23900","25-08-2000 ad 00:00:00","25-09-2000 ad 00:00:00"|||
"4","promotion name# 4","manufacture rebate news","newspaper","8700","18-11-1998 ad 00:00:00","18-01-1999 ad
00:00:00"|||
"5","promotion name# 5","TV program sponsorship","TV","5700","17-03-1999 ad 00:00:00","17-06-1999 ad 00:00:00"|||
"6","promotion name# 6","ad news","newspaper","76800","08-05-2000 ad 00:00:00","08-06-2000 ad 00:00:00"|||
"7","promotion name# 7","online discount","internet","36400","30-06-1998 ad 00:00:00","30-08-1998 ad 00:00:00"|||
```

At the same time a sql*loader controlfile called *SH_PROMOTIONS_25569.ctl* will be generated in SQLLDR_DIR :

```
LOAD DATA
INFILE '/dump/SH_PROMOTIONS_25569.dat' "str X'7c7c7c0a'"
BADFILE '/sqlldr/SH_PROMOTIONS_25569_BAD.dat'
INSERT
INTO TABLE "PROMOTIONS"
FIELDS TERMINATED BY X'2c' ENCLOSED BY X'22'
(
PROMO_ID               DECIMAL EXTERNAL,
PROMO_NAME             CHAR,
PROMO_SUBCATEGORY              CHAR,
PROMO_CATEGORY         CHAR,
PROMO_COST             DECIMAL EXTERNAL,
PROMO_BEGIN_DATE               DATE "DD-MM-YYYY BC HH24:MI:SS",
PROMO_END_DATE         DATE "DD-MM-YYYY BC HH24:MI:SS")
```

See the sql*loader manual to load data into a new database.
Note : if you closely look at INFILE and BADFILE in the controlfile, you'll see that sql*loader will look in /dump for the datafile and /sqlldr for the badfile generation.
This is because of the *MAP_DUMP_DIR, MAP_SQLLDR_DIR* parameters in dude.cfg. (See 3.8 Cross platform unloading parameters)

If *EXPORT_MODE="true"* then an Oracle 8.0.5 DMP compatible file will be generated in *DICTIONARY_DIR*.

10.2 DUMP OBJECTID <table objectid> RESUME AFTER PARTITION <partition objectid>

This command allows you to resume unloading a partitioned table with objectid <table objectid> starting **after** partition with objectid <partition objectid>.
In case a problem occurs while extracting data and you abort the operation or dude halts, you can resume the operation using this command.

10.3 DUMP TABLESPACE

Tablespace per tablespace extraction can be done by using the *DUMP TABLESPACE* command.

```
DUDE> dump tablespace EXAMPLE ;
DUDE> OBJECTID is not a table nor a table partition : 25619
DUDE> OBJECTID is not a table nor a table partition : 25712
DUDE> OBJECTID is not a table nor a table partition : 25618
DUDE> OBJECTID is not a table nor a table partition : 25617
DUDE> OBJECTID is not a table nor a table partition : 25616
DUDE> OBJECTID is not a table nor a table partition : 25615
DUDE> OBJECTID is not a table nor a table partition : 25614
DUDE> OBJECTID is not a table nor a table partition : 25510
DUDE> Found table : OWNER = SH NAME = PRODUCTS
 OBJECTID = 25575 DATA OBJECTID = 25575
COL 1 [1] PROD_ID NUMBER
COL 2 [2] PROD_NAME VARCHAR2 [WE8MSWIN1252]
COL 3 [3] PROD_DESC VARCHAR2 [WE8MSWIN1252]
COL 4 [4] PROD_SUBCATEGORY VARCHAR2 [WE8MSWIN1252]
```

```
COL 5 [5] PROD_SUBCAT_DESC VARCHAR2 [WE8MSWIN1252]
COL 6 [6] PROD_CATEGORY VARCHAR2 [WE8MSWIN1252]
COL 7 [7] PROD_CAT_DESC VARCHAR2 [WE8MSWIN1252]
COL 8 [8] PROD_WEIGHT_CLASS NUMBER
COL 9 [9] PROD_UNIT_OF_MEASURE VARCHAR2 [WE8MSWIN1252]
COL 10 [10] PROD_PACK_SIZE VARCHAR2 [WE8MSWIN1252]
COL 11 [11] SUPPLIER_ID NUMBER
COL 12 [12] PROD_STATUS VARCHAR2 [WE8MSWIN1252]
COL 13 [13] PROD_LIST_PRICE NUMBER
COL 14 [14] PROD_MIN_PRICE NUMBER
start dumping ...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows    = 10000
Migrated rows  = 0
Chained rows   = 0
Deleted rows    = 0 (not extracted!)
Skipped rows   = 0 (not extracted!)

DUDE> Done !
DUDE> Found table : OWNER = SH NAME = CUSTOMERS
 OBJECTID = 25573 DATA OBJECTID = 25573
COL 1 [1] CUST_ID NUMBER
COL 2 [2] CUST_FIRST_NAME VARCHAR2 [WE8MSWIN1252]
COL 3 [3] CUST_LAST_NAME VARCHAR2 [WE8MSWIN1252]
COL 4 [4] CUST_GENDER CHAR [WE8MSWIN1252]
COL 5 [5] CUST_YEAR_OF_BIRTH NUMBER
COL 6 [6] CUST_MARITAL_STATUS VARCHAR2 [WE8MSWIN1252]
COL 7 [7] CUST_STREET_ADDRESS VARCHAR2 [WE8MSWIN1252]
COL 8 [8] CUST_POSTAL_CODE VARCHAR2 [WE8MSWIN1252]
COL 9 [9] CUST_CITY VARCHAR2 [WE8MSWIN1252]
COL 10 [10] CUST_STATE_PROVINCE VARCHAR2 [WE8MSWIN1252]
COL 11 [11] COUNTRY_ID CHAR [WE8MSWIN1252]
COL 12 [12] CUST_MAIN_PHONE_NUMBER VARCHAR2 [WE8MSWIN1252]
COL 13 [13] CUST_INCOME_LEVEL VARCHAR2 [WE8MSWIN1252]
COL 14 [14] CUST_CREDIT_LIMIT NUMBER
COL 15 [15] CUST_EMAIL VARCHAR2 [WE8MSWIN1252]
start dumping ...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows    = 50000
Migrated rows  = 0
Chained rows   = 0
Deleted rows    = 0 (not extracted!)
Skipped rows   = 0 (not extracted!)

DUDE> Done !
DUDE> OBJECTID is not a table nor a table partition : 25572
DUDE> Found table : OWNER = SH NAME = COUNTRIES
 OBJECTID = 25571 DATA OBJECTID = 25571
COL 1 [1] COUNTRY_ID CHAR [WE8MSWIN1252]
COL 2 [2] COUNTRY_NAME VARCHAR2 [WE8MSWIN1252]
COL 3 [3] COUNTRY_SUBREGION VARCHAR2 [WE8MSWIN1252]
COL 4 [4] COUNTRY_REGION VARCHAR2 [WE8MSWIN1252]
start dumping ...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows    = 19
Migrated rows  = 0
Chained rows   = 0
Deleted rows    = 0 (not extracted!)
Skipped rows   = 0 (not extracted!)

DUDE> Done !
DUDE> OBJECTID is not a table nor a table partition : 25609
DUDE> OBJECTID is not a table nor a table partition : 25608
```

```
DUDE> OBJECTID is not a table nor a table partition : 25607
DUDE> Found table partition : NAME = COSTS_Q4_2000 OWNER = SH TABLE = COSTS OBJ#
 = 25606 DATAOBJ# = 25606 BO# = 25594
start dumping ...
DUDE> Reading blocks done !
..........
```

10.4 DUMP TABLESPACE <tablespace_name> RESUME AFTER <objectid>

In case a problem occurs while extracting data and you abort the operation, you can later resume the operation using this command.
This command allows you to resume unloading *after* the specified OBJECTID.The following table has OBJECTID 17001. To resume your tablespace unloading operation, just start DUDE again and issue the command :

```
DUDE> dump tablespace EXAMPLE resume after 17001 ;
```

Basically, it'll skip all objects including the one specified.

10.5 DUMP USER

It is possible to dump entire schema's. Note that a blockmap must exist for every tablespace/datafile containing a segment of the specified schema.

```
DUDE> dump user SH ;
```

A list of all users can be created using the *SHOW USERS* command.

10.6 DUMP USER <user> RESUME AFTER <objectid>

In case a problem occurs while extracting data and you abort the operation or dude halts, you can later resume the operation using this command.
This command allows you to resume unloading *after* the specified OBJECTID.
To resume your schema unloading operation, just start DUDE again and issue the command :

```
DUDE> dump user SCOTT resume after 17001 ;
```

Basically, it'll skip all objects including the one specified, and then continue unloading, starting with the next objectid of that user.

10.7 DUMP USER <user> RESUME AFTER <table objectid> PARTITION <partition objectid>

In case a problem occurs while extracting data and you abort the operation or dude halts, you can later resume the operation using this command.
This command allows you to resume unloading *after* the specified OBJECTID and after partition with objectid <partition objectid>.
This command was introduced for very large partitioned tables.
To resume your schema unloading operation, just start DUDE again and issue the command :

```
DUDE> dump user SCOTT resume after 17001 partition 17004 ;
```

10.8 DUMP TABLE

Instead of dumping a table by objectid, it is possible to dump a table by owner and name.

```
DUDE> dump table SH TIMES ;
```

10.9 Empty tables

If your table did not contain any data, the blockmapper will not pick up any blocks for this table. This is because the blockmapper only checks for datablocks and not for segment headers. This means that

IF YOUR TABLE DOES NOT CONTAIN DATA, THERE WILL BE NO DUMP FILE

10.10 Table types

DUDE will support a whole range of tables for unloading, this includes normal heap tables, partitioned and subpartitoned head tables, index organized tables, partitioned index organized tables (with or without overflow segments).

If no SYSTEM tablespace is available, the heuristic scanner will **not** pick up IOT's, and thus, without a SYSTEM tablespace you will not be able to unload IOT's !

11 Generating DDL

11.1 GENERATE DDL FOR

Table DDL is automatically generated when a table is unloaded if parameter *GEN_TABLE_DDL* equals *true*. However, in some cases it might be necessary to regenerated the table ddl without unloading the actual data.
This can be done with done with the following commands :

```
DUDE> generate ddl for tablespace TABLESPACE_NAME  ;
```
Generates ddl for all tables in tablespace TABLESPACE_NAME

```
DUDE> generate ddl for table  OWNER TABLENAME ;
```
Generates ddl for a specific table specified by owner and tablename

```
DUDE> generate ddl for ojbectid 12200 ;
```
Generates ddl for a specific table specified by objectid

Note - DDL for partitioned and sub partitioned tables is generated as DDL for the base table (without partition clause)
Note – DDL for index organized tables is generated as DDL for the base table (as a normal heap table)

11.2 GENERATE INDEX DDL

If the *SYSTEM* tablespace is still available, the *GENERATE INDEX DDL* will enable you to generate the DDL for all indexes.
The index ddl is generated in separate files located in the directory *DDL_DIR*. The file format is *<TABLE OWNER>_<TABLE NAME>_<INDEX NAME>_INDEX.sql*

```
DUDE> generate index ddl ;
```

11.3 GENERATE SQUENCE DDL

If the *SYSTEM* tablespace is still available, the *GENERATE SEQUENCE DDL* will enable you to generate the DDL for sequences.
The sequence ddl will be grouped by owner in separate files located in the directory *DDL_DIR*. The file format is *SEQ_<OWNER>.sql*

```
DUDE> generate sequence ddl ;
```

11.4 GENERATE VIEW DDL

If the *SYSTEM* tablespace is still available, the *GENERATE VIEW DDL* will enable you to generate the DDL for views.
The view ddl will be grouped by owner in separate files located in the directory *DDL_DIR*. The file format is *VIEW_<OWNER>.sql*

```
DUDE> generate view ddl ;
```

11.5 Dumping PLSQL code

Using the *DUMP PLSQL* command, all PLSQL code will be extracted, sorted by user, name, type and finally sorted by line number. The extracted PLSQL code is placed in *DICTIONARY_DIR* directory, in the file PLSQL.dat.
Once the extraction is completed – the PLSQL.dat file is parsed and sorted by user, name, type and line number in the *PLSQL_DIR* directory.
This routine will spawn error messages when obfuscated code is handled – this will also result in faulty *.tmp and *.sql files in *PLSQL_DIR*.
DUDE will skip PLSQL code owned by SYS.

```
DUDE> dump PLSQL ;
DUDE> Dumping PLSQL code ...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows     = 87941
Migrated rows   = 0
Chained rows    = 0
Deleted rows     = 2130 (not extracted!)
Skipped rows    = 0 (not extracted!)

DUDE> Done !
DUDE> Please wait ... sorting dump !...
01-07-2005 17:01 : DUDE> Sorting to :
c:\oracle_stuff\dude_plsql\OE_13_PHONE_LIST_TYP.sql
01-07-2005 17:01 : DUDE> Processing file :
c:\oracle_stuff\dude_plsql\OE_13_PRODUCT_INFORMATION_TYP.tmp
01-07-2005 17:01 : DUDE> Sorting to :
c:\oracle_stuff\dude_plsql\OE_13_PRODUCT_INFORMATION_TYP.sql
01-07-2005 17:01 : DUDE> Processing file :
c:\oracle_stuff\dude_plsql\OE_13_PRODUCT_REF_LIST_TYP.tmp
01-07-2005 17:01 : DUDE> Sorting to :
c:\oracle_stuff\dude_plsql\OE_13_PRODUCT_REF_LIST_TYP.sql
01-07-2005 17:01 : DUDE> Processing file :
c:\oracle_stuff\dude_plsql\OE_13_SUBCATEGORY_REF_LIST_TYP.tmp
01-07-2005 17:01 : DUDE> Sorting to :
c:\oracle_stuff\dude_plsql\OE_13_SUBCATEGORY_REF_LIST_TYP.sql
01-07-2005 17:01 : DUDE> Processing file :
c:\oracle_stuff\dude_plsql\OE_13_WAREHOUSE_TYP.tmp
01-07-2005 17:01 : DUDE> Sorting to : c:\oracle_stuff\dude_plsql\OE_13_WAREHOUSE_TYP.sql
01-07-2005 17:01 : DUDE> Processing file :
c:\oracle_stuff\dude_plsql\OE_14_CATALOG_TYP.tmp
01-07-2005 17:01 : DUDE> Sorting to : c:\oracle_stuff\dude_plsql\OE_14_CATALOG_TYP.sql
01-07-2005 17:01 : DUDE> Processing file :
c:\oracle_stuff\dude_plsql\OE_14_COMPOSITE_CATEGORY_TYP.tmp
```

```
01-07-2005 17:01 : DUDE> Sorting to :
c:\oracle_stuff\dude_plsql\OE_14_COMPOSITE_CATEGORY_TYP.sql
01-07-2005 17:01 : DUDE> Processing file :
c:\oracle_stuff\dude_plsql\OE_14_LEAF_CATEGORY_TYP.tmp
01-07-2005 17:01 : DUDE> Sorting to :
c:\oracle_stuff\dude_plsql\OE_14_LEAF_CATEGORY_TYP.sql
01-07-2005 17:01 : DUDE> Processing file :
c:\oracle_stuff\dude_plsql\OUTLN_7_ORA$GRANT_SYS_SELECT.tmp
01-07-2005 17:01 : DUDE> Sorting to :
c:\oracle_stuff\dude_plsql\OUTLN_7_ORA$GRANT_SYS_SELECT.sql
01-07-2005 17:01 : DUDE> Processing file :
c:\oracle_stuff\dude_plsql\PERFSTAT_11_STATSPACK.tmp
01-07-2005 17:01 : DUDE> Sorting to :
c:\oracle_stuff\dude_plsql\PERFSTAT_11_STATSPACK.sql
01-07-2005 17:01 : DUDE> Processing file :
c:\oracle_stuff\dude_plsql\PERFSTAT_7_POPULATE_REP.tmp
01-07-2005 17:01 : DUDE> Sorting to :
c:\oracle_stuff\dude_plsql\PERFSTAT_7_POPULATE_REP.sql
01-07-2005 17:01 : DUDE> Processing file :
c:\oracle_stuff\dude_plsql\PERFSTAT_7_SEGMENT_GROWTH.tmp
01-07-2005 17:01 : DUDE> Sorting to :
c:\oracle_stuff\dude_plsql\PERFSTAT_7_SEGMENT_GROWTH.sql
01-07-2005 17:01 : DUDE> Processing file :
c:\oracle_stuff\dude_plsql\PERFSTAT_9_STATSPACK.tmp
01-07-2005 17:01 : DUDE> Sorting to :
c:\oracle_stuff\dude_plsql\PERFSTAT_9_STATSPACK.sql
...
Errors due to PLSQL code obfuscation :
01-07-2005 17:01 : DUDE> OBJECTID = 8710 : error retrieving object details
01-07-2005 17:01 : DUDE> SYS_0__NEXT_OBJECT.tmp : error on line : 135
01-07-2005 17:01 : DUDE> OBJECTID = 8710 : error retrieving object details
01-07-2005 17:01 : DUDE> SYS_0__NEXT_OBJECT.tmp : error on line : 136
01-07-2005 17:01 : DUDE> OBJECTID = 8710 : error retrieving object details
01-07-2005 17:01 : DUDE> SYS_0__NEXT_OBJECT.tmp : error on line : 137
01-07-2005 17:01 : DUDE> OBJECTID = 8710 : error retrieving object details
01-07-2005 17:01 : DUDE> SYS_0__NEXT_OBJECT.tmp : error on line : 138
01-07-2005 17:01 : DUDE> OBJECTID = 8710 : error retrieving object details
01-07-2005 17:01 : DUDE> SYS_0__NEXT_OBJECT.tmp : error on line : 139
01-07-2005 17:01 : DUDE> OBJECTID = 8710 : error retrieving object details
01-07-2005 17:01 : DUDE> SYS_0__NEXT_OBJECT.tmp : error on line : 141
01-07-2005 17:01 : DUDE> OBJECTID = 8710 : error retrieving object details
01-07-2005 17:01 : DUDE> SYS_0__NEXT_OBJECT.tmp : error on line : 149
01-07-2005 17:01 : DUDE> OBJECTID = 8710 : error retrieving object details
01-07-2005 17:01 : DUDE> SYS_0__NEXT_OBJECT.tmp : error on line : 152
01-07-2005 17:01 : DUDE> OBJECTID = 8710 : error retrieving object details
01-07-2005 17:01 : DUDE> SYS_0__NEXT_OBJECT.tmp : error on line : 156
01-07-2005 17:01 : DUDE> OBJECTID = 8710 : error retrieving object details
01-07-2005 17:01 : DUDE> SYS_0__NEXT_OBJECT.tmp : error on line : 200
01-07-2005 17:01 : DUDE> OBJECTID = 8710 : error retrieving object details
01-07-2005 17:01 : DUDE> SYS_0__NEXT_OBJECT.tmp : error on line : 220
01-07-2005 17:01 : DUDE> OBJECTID = 8710 : error retrieving object details
01-07-2005 17:01 : DUDE> SYS_0__NEXT_OBJECT.tmp : error on line : 229
01-07-2005 17:01 : DUDE> OBJECTID = 8710 : error retrieving object details
```

12 No SYSTEM, no problem

12.1 Is SYSTEM really gone ?

What to do when you've lost your SYSTEM tablespaces. A SYSTEM tablespace is lost when no copy of the system datafile, related to the target database, is available. A system datafile is related to the target database if it contains **all or most** table definitions (object id's must match) related to the target database.
So this does not mean that SYSTEM should be consistent with the target database !

Example 1
Target database crashed/corrupted – all tablespaces/datafiles are available from backup media except SYSTEM.
The DBA managed to find a 3 months old copy somewhere, yet no way to recover it due to lack of archives.

This is a related system tablespace as it once belonged to the target database. If there were few tables dropped, (re)created or moved then this SYSTEM tablespace gives us an excellent chance for recovery.

Example 2
Target database crashed/corrupted – all tablespaces/datafiles are available from backup media except SYSTEM. The test database is a clone from production. The clone was created by restoring and recovering a (hot) backup from production.

The SYSTEM tablespace of the test database is a related SYSTEM tablespace. If very few tables where dropped, (re)created and/or moved, all object id's should relate to the tables found in the tablespaces.

Example 3
Target database crashed/corrupted – all tablespaces/datafiles are available from backup media except SYSTEM. The test database is a clone from production. However, the clone is a newly created database – data was imported using Oracle exp/imp utilities.

The SYSTEM tablespace of the test database is not related to production. There's no guarantee that the object id's found in the dictionary of the test database match the object id's found in the production database.

12.2 Getting started

12.2.1 Blockmaps

As always – before doing anything else – create blockmaps of the available tablespaces. (See *4. Blockmaps*)
Remember – once you have created a blockmap for a tablespace, you do not have to do this again.

```
DUDE> create blockmap for tablespace EXAMPLE ;
DUDE> ID := 0 BLOCKMAPPER for TABLESPACE NAME = EXAMPLE
OFFSET = 0
ASSM = true
BIGFILE = false
BLOCKSIZE = 8192
NUMBER = 3 FILENAME = C:\ORACLE\ORADATA\KVMB\EXAMPLE01.DBF IBS = 8192


DUDE> Datafile C:\ORACLE\ORADATA\KVMB\EXAMPLE01.DBF : start reading blocks ...
DUDE> Datafile C:\ORACLE\ORADATA\KVMB\EXAMPLE01.DBF : reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Please wait for chunk stream to finish ...
DUDE> Done !
```

12.2.2 Scan data

Because no dictionary is available, there's no way to determine table names, column names, column datatypes, number of columns, column order and so on. There is **no way** table names and column names can be retrieved. However, we can try to determine the number of columns and their datatypes.
To do this, we'll have to analyze the data using the *SCAN OBJECTID* and/or *SCAN TABLESPACE* commands.
The parameters *SAMPLE_LIMIT* and *SAMPLE_SIZE* define how many rows will be scanned per object.

12.2.2.1 SCAN OBJECTID

SCAN OBJECTID analyzes the specified <objectid>.
For each <objectid> a file is created in the directory *SCAN_DIR* containing the command to unload the object. The name of this file is *SCAN_<objectid>.dde*. We will further refer to this file as a 'dude' file.

```
DUDE> scan objectid <objectid> ;
```

12.2.2.2 SCAN TABLESPACE

SCAN TABLESPACE analyzes the entire tablespace specified by <tablespace_name>. For each <objectid> found in the tablespace a dude file is created.

```
DUDE> scan tablespace <tablespace_name> ;

DUDE> scan tablespace EXAMPLE ;
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
No rowstats available ! Rows sampled : 0

DUDE> Done !
DUDE> Scanning OBJECTID : 25575
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
Rows sampled : 3030
COL 0 : LEN = 4 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 37% : NICE NUM = 100%
COL 1 : LEN = 50 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 2%
COL 2 : LEN = 127 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 0%
COL 3 : LEN = 26 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 0%
COL 4 : LEN = 46 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 0%
COL 5 : LEN = 5 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 43%
COL 6 : LEN = 25 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 0%
COL 7 : LEN = 2 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 0% : NICE NUM = 100%
COL 8 : LEN = 1 : NULLS = 100% : POSS. DATE = 0% : PRINTABLE = 0% : NICE NUM = 0%
COL 9 : LEN = 16 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 0%
COL 10 : LEN = 3 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 21% : NICE NUM = 100%
COL 11 : LEN = 19 : NULLS = 0% : POSS. DATE = 5% : PRINTABLE = 100% : NICE NUM = 0%
COL 12 : LEN = 4 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 40% : NICE NUM = 100%
COL 13 : LEN = 4 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 37% : NICE NUM = 100%

DUDE> Done !
DUDE> Scanning OBJECTID : 25573
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
Rows sampled : 5211
COL 0 : LEN = 4 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 34% : NICE NUM = 100%
COL 1 : LEN = 11 : NULLS = 0% : POSS. DATE = 19% : PRINTABLE = 100% : NICE NUM = 30%
COL 2 : LEN = 11 : NULLS = 0% : POSS. DATE = 18% : PRINTABLE = 100% : NICE NUM = 28%
COL 3 : LEN = 1 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 32%
COL 4 : LEN = 3 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 32% : NICE NUM = 100%
COL 5 : LEN = 7 : NULLS = 35% : POSS. DATE = 44% : PRINTABLE = 99% : NICE NUM = 0%
COL 6 : LEN = 32 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 56%
COL 7 : LEN = 5 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 100%
COL 8 : LEN = 23 : NULLS = 0% : POSS. DATE = 12% : PRINTABLE = 100% : NICE NUM = 27%
COL 9 : LEN = 30 : NULLS = 0% : POSS. DATE = 4% : PRINTABLE = 100% : NICE NUM = 23%
COL 10 : LEN = 2 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 31%
COL 11 : LEN = 12 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 100%
COL 12 : LEN = 20 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 4%
COL 13 : LEN = 3 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 23% : NICE NUM = 100%
COL 14 : LEN = 23 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 0%

DUDE> Done !
DUDE> Scanning OBJECTID : 25572
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
No rowstats available ! Rows sampled : 0
```

```
DUDE> Done !
DUDE> Scanning OBJECTID : 25571
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
Rows sampled : 19
COL 0 : LEN = 2 : NULLS = 0% : POSS. DATE = 5% : PRINTABLE = 100% : NICE NUM = 36%
COL 1 : LEN = 24 : NULLS = 0% : POSS. DATE = 21% : PRINTABLE = 100% : NICE NUM = 26%
COL 2 : LEN = 16 : NULLS = 0% : POSS. DATE = 5% : PRINTABLE = 100% : NICE NUM = 31%
COL 3 : LEN = 11 : NULLS = 0% : POSS. DATE = 15% : PRINTABLE = 100% : NICE NUM = 84%

DUDE> Done !
DUDE> Scanning OBJECTID : 25609
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
Rows sampled : 3936
COL 0 : LEN = 4 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 39% : NICE NUM = 100%
COL 1 : LEN = 11 : NULLS = 0% : POSS. DATE = 1% : PRINTABLE = 40% : NICE NUM = 99%
COL 2 : LEN = 11 : NULLS = 0% : POSS. DATE = 95% : PRINTABLE = 18% : NICE NUM = 1%
COL 3 : LEN = 1 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 13%
COL 4 : LEN = 3 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 52% : NICE NUM = 100%
COL 5 : LEN = 7 : NULLS = 1% : POSS. DATE = 1% : PRINTABLE = 14% : NICE NUM = 96%
COL 6 : LEN = 32 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 50% : NICE NUM = 96%
COL 7 : LEN = 5 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 5%
COL 8 : LEN = 21 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 1%
COL 9 : LEN = 30 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 1%
COL 10 : LEN = 2 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 2%
COL 11 : LEN = 12 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 5%
COL 12 : LEN = 20 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 0%
COL 13 : LEN = 3 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 24% : NICE NUM = 5%
COL 14 : LEN = 23 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 0%

DUDE> Done !
DUDE> Scanning OBJECTID : 25608
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
Rows sampled : 19129
COL 0 : LEN = 4 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 36% : NICE NUM = 100%
COL 1 : LEN = 4 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 34% : NICE NUM = 100%
COL 2 : LEN = 7 : NULLS = 0% : POSS. DATE = 100% : PRINTABLE = 14% : NICE NUM = 0%
COL 3 : LEN = 1 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM = 11%
COL 4 : LEN = 3 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 54% : NICE NUM = 100%
COL 5 : LEN = 2 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 4% : NICE NUM = 100%
COL 6 : LEN = 4 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 29% : NICE NUM = 100%

DUDE> Done !
DUDE> Scanning OBJECTID : 25607
…
```

This results in the creation of *DUDE dde* files in the *SCAN_DIR* directory. The filename is always ***SCAN_<dataobjectid>.dde***.

SCAN_25575.dde contains :

```
dump SCANNED OBJECTID 25575 (  COL0 NUMBER ,  COL1 CHAR ,  COL2 CHAR ,  COL3 CHAR ,
COL4 CHAR ,  COL5 CHAR ,  COL6 CHAR ,  COL7 NUMBER ,  COL8 CHAR ,  COL9 CHAR ,  COL10
NUMBER ,  COL11 CHAR ,  COL12 NUMBER ,  COL13 NUMBER ) ;
```

12.2.2.3 SCAN TABLESPACE ORPHANED

SCAN TABLESPACE ORPHANED analyzes the entire tablespace specified by <tablespace_name>, however it will **skip objects found in the datadictionary** – only orphaned segments/blocks are scanned. Contrary to the original *SCAN TABLESPACE* command, a **datadictionary must be available**.
For each orphaned <objectid> found in the tablespace a dude file is created.

```
DUDE> scan tablespace <tablespace_name> ORPHANED ;
```

12.2.3 Extracting scanned data

Extracting scanned data is done by executing the commands found in the dude files. There are two ways to do this :

- object by object using the *DUMP SCANNED OBJECTID <objectid>* command as found in the dude dde file (copy/paste command)
- tablespace by tablespace using the *DUMP SCANNED TABLESPACE <tablespace_name>* command which basically loads all dude files related to <tablespace_name> into DUDE.

12.2.3.1 DUMP SCANNED OBJECTID

This statement is basically the same as *DUMP OBJECTID* – however you must specify all columns and their datatypes.
Valid datatype keywords are NUMBER, CHAR, DATE, LONG, TIMESTAMP, BINARY_FLOAT and BINARY_DOUBLE.

Example :

```
DUDE> dump SCANNED OBJECTID 25575 (  COL0 NUMBER ,  COL1 CHAR ,  COL2 CHAR ,  COL3 CHAR
,  COL4 CHAR ,  COL5 CHAR ,  COL6 CHAR ,  COL7 NUMBER ,  COL8 CHAR ,  COL9 CHAR ,  COL10
NUMBER ,  COL11 CHAR ,  COL12 NUMBER ,  COL13 NUMBER ) ;
DUDE> Dumping objectid 25575
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows     = 10000
Migrated rows   = 0
Chained rows    = 0
Deleted rows     = 0 (not extracted!)
Skipped rows    = 0 (not extracted!)

DUDE> Done !
```

```
DUDE> Dumping objectid 25575 Done !
```

TIP : if you set your *SCRIPT_DIR* equal to your *SCAN_DIR* you can easily extract the object without copy/pasting the command by using the *START* command.
Example :

```
DUDE> start SCAN_25575.dde ;
DUDE> dump SCANNED OBJECTID 25575 (  COL0 NUMBER ,  COL1 CHAR ,  COL2 CHAR ,  CO
L3 CHAR ,  COL4 CHAR ,  COL5 CHAR ,  COL6 CHAR ,  COL7 NUMBER ,  COL8 CHAR ,  CO
L9 CHAR ,  COL10 NUMBER ,  COL11 CHAR ,  COL12 NUMBER ,  COL13 NUMBER ) ;
DUDE> Dumping objectid 25575
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows    = 10000
Migrated rows  = 0
Chained rows   = 0
Deleted rows    = 0 (not extracted!)
Skipped rows   = 0 (not extracted!)

DUDE> Done !
DUDE> Dumping objectid 25575 Done !
```

12.2.3.2 DUMP SCANNED TABLESPACE

This command reads all dude files related to the tablespace into DUDE and executes them.

12.3 Scanning issues and limitations

When the dictionary is unavailable, it is a tedious task to map the data to the correct tables and columns. Hopefully you'll have one of database developers at your side to do this.
As you will read in the following section, there are some issues that make this task an even harder challenge.

12.3.1 Dropped tables

What happens if a table is dropped in Oracle ? Well – among others, records are deleted in obj$, tab$ and col$. The datablocks itself are not removed. In other words, only the table's metadata will be deleted.
This means that a dropped table will be seen by the blockmapper routine – and blocks will be analyzed by the *SCAN TABLESPACE* command.
You'll have to keep this in mind while mapping your data to your tables.

Example
Table X was dropped. Later, it was recreated.
While examing data extracted by DUDE, we find 2 flatfiles that look like data from table X : SCAN_8900.dat and SCAN_9112.dat. How can we be sure we have the most recent data ?

The name of the flatfile contains the objectid. The flatfile with the highest objectid is the most recent. SCAN_8900.dat will contain data from the dropped table X – SCAN_9112.dat of the newly created table X.

12.3.2 Non-heap tables

The heuristic algorithm for detecting columns and datatypes is only capable of looking into blocks belonging to normal heap tables. It can not correctly identify columns in clustered tables, compressed tables and index organized tables (IOT).

12.3.3 Trailing NULL columns

Trailing NULL columns are not stored internally. This is no problem if the Oracle dictionary is intact, because then Oracle knows how many columns the table consists of.
If the dictionary is lost, there's no way to find out the exact number of columns if a table's trailing columns are all null. Keep this in mind while mapping your data to tables.

12.3.4 Wrong datatype

Datatypes are resolved through a number of rules – however – you should not blindly rely on these results. It is after all a best effort guess.
If you think a wrong datatype was chosen to unload data, then you can edit the corresponding dude file and fill in the correct datatype manually.
Since DUDE version 2.3, the heuristic scanning routine has been optimized. The new routine takes into account the *niceness* of numbers. This results in extremely good datatype recognition.

Example :

```
DUDE> scan objectid 8977 ;
Reading blocks done !
DUDE> Number of rows skipped : 0
DUDE>
Rows sampled : 2183
COL 0 : LENGTH = 3 : NULLS = 0% : POSSIBLE DATE = 0% : PRINTABLE = 89%
COL 1 : LENGTH = 3 : NULLS = 57% : POSSIBLE DATE = 0% : PRINTABLE = 89%
COL 2 : LENGTH = 2 : NULLS = 0% : POSSIBLE DATE = 0% : PRINTABLE = 67%
COL 3 : LENGTH = 30 : NULLS = 0% : POSSIBLE DATE = 1% : PRINTABLE = 100%
COL 4 : LENGTH = 2 : NULLS = 0% : POSSIBLE DATE = 0% : PRINTABLE = 50%
COL 5 : LENGTH = 13 : NULLS = 92% : POSSIBLE DATE = 0% : PRINTABLE = 99%
COL 6 : LENGTH = 2 : NULLS = 0% : POSSIBLE DATE = 0% : PRINTABLE = 50%
COL 7 : LENGTH = 7 : NULLS = 0% : POSSIBLE DATE = 100% : PRINTABLE = 42%
COL 8 : LENGTH = 7 : NULLS = 0% : POSSIBLE DATE = 100% : PRINTABLE = 42%
COL 9 : LENGTH = 7 : NULLS = 0% : POSSIBLE DATE = 100% : PRINTABLE = 42%
COL 10 : LENGTH = 2 : NULLS = 0% : POSSIBLE DATE = 0% : PRINTABLE = 50%
COL 11 : LENGTH = 6 : NULLS = 99% : POSSIBLE DATE = 0% : PRINTABLE = 92%
COL 12 : LENGTH = 5 : NULLS = 99% : POSSIBLE DATE = 0% : PRINTABLE = 90%
COL 13 : LENGTH = 2 : NULLS = 0% : POSSIBLE DATE = 0% : PRINTABLE = 4%
COL 14 : LENGTH = 16 : NULLS = 96% : POSSIBLE DATE = 0% : PRINTABLE = 64%
COL 15 : LENGTH = 2 : NULLS = 0% : POSSIBLE DATE = 0% : PRINTABLE = 47%
COL 16 : LENGTH = 4 : NULLS = 0% : POSSIBLE DATE = 0% : PRINTABLE = 72%

DUDE> dump SCANNED OBJECTID 8977 ( COL0 NUMBER , COL1 NUMBER , COL2 NUMBER , COL3 CHAR ,
COL4 NUMBER , COL5 CHAR , COL6 NUMBER , COL7 DATE , COL8 DATE , COL9 DATE , COL10 NUMBER
, COL11 NUMBER , COL12 NUMBER , COL13 NUMBER , COL14 NUMBER , COL15 NUMBER , COL16
NUMBER ) ;
DUDE> Dumping objectid 8977
Reading blocks done !
DUDE> Number of rows skipped : 0
DUDE> Dumping objectid 8977 Done !
```

DUDE_8977.dat :

```
"8422","","51","CATALOG_TYP","1","$VSN_1","13","18-06-2005 ad 17:00:01","18-06-2005 ad 17:00:01","18-06-2005 ad
17:00:01","1","","","0","-0.000000000000000000000000000-1030-1100-1230336670-56500-50650-60-1120-106","0","65535"
"8438","","51","CATALOG_TYP","1","","13","18-06-2005 ad 17:00:03","18-06-2005 ad 17:00:03","18-06-2005 ad
17:00:03","1","","","0","0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000001631741057522510010822351070-13122323760","6","65535"
"8420","","51","COMPOSITE_CATEGORY_TYP","1","$VSN_1","13","18-06-2005 ad 17:00:01","18-06-2005 ad 17:00:01","18-06-
2005 ad
17:00:01","1","","","0","15518618219515172166137451559743168242100000000000000000000000000000000000000000000000000000
000000000000000000","0","65535"
"8437","","51","COMPOSITE_CATEGORY_TYP","1","","13","18-06-2005 ad 17:00:03","18-06-2005 ad 17:00:03","18-06-2005 ad
17:00:03","1","","","0","0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000001631741057522410010822351070-13122323760","6","65535"
"8418","","51","LEAF_CATEGORY_TYP","1","$VSN_1","13","18-06-2005 ad 17:00:00","18-06-2005 ad 17:00:00","18-06-2005 ad
17:00:00","1","","","0","-
0.0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000-205463740-49230-
1040-400-9468830-578159","0","65535"
```

```
"8436","","51","LEAF_CATEGORY_TYP","1","","13","18-06-2005 ad 17:00:02","18-06-2005 ad 17:00:02","18-06-2005 ad
17:00:02","1","","","0","0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000001631741057522310010822351070-13122323760","6","65535"
"8414","","51","CATEGORY_TYP","1","$VSN_1","13","18-06-2005 ad 17:00:00","18-06-2005 ad 17:00:00","18-06-2005 ad
17:00:00","1","","","0","0.0000000000000000000000000000000000204178036192710-1178134963187248241122","0","65535"
"8435","","51","CATEGORY_TYP","1","","13","18-06-2005 ad 17:00:02","18-06-2005 ad 17:00:02","18-06-2005 ad
17:00:02","1","","","0","0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000001631741057521910010822351070-13122323760","6","65535"
"8439","","0","MEDIA_DIR","9","","23","18-06-2005 ad 17:00:05","18-06-2005 ad 17:00:05","18-06-2005 ad
17:00:05","1","","","0","","6","65535"
"8440","","52","ADHEADER_TYP","1","","13","18-06-2005 ad 17:00:05","18-06-2005 ad 17:00:05","18-06-2005 ad
17:00:05","1","","","0","0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000001631741057520510010822351070-13122323760","6","65535"
"8441","","52","TEXTDOC_TYP","1","","13","18-06-2005 ad 17:00:05","18-06-2005 ad 17:00:05","18-06-2005 ad
17:00:05","1","","","0","0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000001631741057520610010822351070-13122323760","6","65535"
"8442","","52","TEXTDOC_TAB","1","","13","18-06-2005 ad 17:00:05","18-06-2005 ad 17:00:05","18-06-2005 ad
17:00:05","1","","","0","0.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000001631741057520710010822351070-13122323760","6","65535"
"8443","8443","52","PRINT_MEDIA","1","","2","18-06-2005 ad 17:00:05","18-06-2005 ad 17:00:05","18-06-2005 ad
17:00:05","1","","","0","","6","1"
```

Allthough maybe unclear, you can see that the 15th (or COL14) column is extract as a NUMBER. However, this results in rather weird numbers. Most likely this was not a NUMBER. Edit the *DUMP SCANNED OBJECTID* command to reflect the correct datatype.

```
DUDE> dump SCANNED OBJECTID 8977 ( COL0 NUMBER , COL1 NUMBER , COL2 NUMBER , COL3 CHAR ,
COL4 NUMBER , COL5 CHAR , COL6 NUMBER , COL7 DATE , COL8 DATE , COL9 DATE , COL10 NUMBER
, COL11 NUMBER , COL12 NUMBER , COL13 NUMBER , COL14 CHAR , COL15 NUMBER , COL16 NUMBER
) ;
DUDE> Dumping objectid 8977
Reading blocks done !
DUDE> Number of rows skipped : 0
DUDE> Dumping objectid 8977 Done !
```

12.3.5 Supported datatypes

The scan algorithm can only detect the following datatypes :

- DATE
- VARCHAR2/CHAR/LONG
- NUMBER

Allthough not detected by the algorithm, the following datatypes are also supported in the *DUMP SCANNED* syntax :

- TIMESTAMP
- BINARY_FLOAT
- BINARY_DOUBLE

Not support in case of missing SYSTEM tablespace :

- BLOB
- CLOB

12.3.6 Missing tables

If a table did not contain any rows, it will not be detected by the block scanner.

12.3.7 Internal column order

Imagine table X (a DATE, b VARCHAR(2000), c NUMBER, d NUMBER). Without going into details – it is quite possible that Oracle will store the columns physically as a, c, d and then b. Again, keep this in mind while identifying tables.

13 Miscellaneous

13.1 START <dude script> command

START <dude script> runs the statements in the specified DUDE script.

If *<dude script>* contains a path, then *SCRIPT_DIR* is not evaluated. Otherwise, *<dude script>* is searched for in *SCRIPT_DIR*.

If *SCRIPT_DIR* is not defined in *dude.cfg*, *SCRIPT_DIR* defaults to the *current* directory (directory DUDE was started from).

Example:

Contents of *c:\oracle_stuff\dude\start.dde* :
drop blockmap for tablespace TS_DATA ;
drop blockmap for tablespace SYSTEM ;
create blockmap for tablespace SYSTEM ;
create blockmap for tablespace TS_DATA ;
dump dictionary ;
create dictionary ;
show users ;
exit ;

```
DUDE> start c:\oracle_stuff\dude\start.dde
DUDE> ;
DUDE> drop blockmap for tablespace TS_DATA ;
DUDE> drop blockmap for tablespace SYSTEM ;
DUDE> create blockmap for tablespace SYSTEM ;
DUDE> ID := 0 BLOCKMAPPER for TABLESPACE NAME = SYSTEM
OFFSET = 0
ASSM = false
BLOCKSIZE = 8192
NUMBER = 1 FILENAME = f:\10g\SYSTEM01.DBF IBS = 8192


DUDE> Datafile f:\10g\SYSTEM01.DBF : start reading blocks ...
DUDE> Datafile f:\10g\SYSTEM01.DBF : reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Done !
DUDE> create blockmap for tablespace TS_DATA ;
DUDE> ID := 0 BLOCKMAPPER for TABLESPACE NAME = TS_DATA
OFFSET = 0
ASSM = true
BLOCKSIZE = 8192
NUMBER = 6 FILENAME = f:\10g\ts_data_01.DBF IBS = 8192
NUMBER = 7 FILENAME = f:\10g\ts_data_02.DBF IBS = 8192


DUDE> Datafile f:\10g\ts_data_01.DBF : start reading blocks ...
DUDE> Datafile f:\10g\ts_data_01.DBF : reading blocks done !
DUDE> Datafile f:\10g\ts_data_02.DBF : start reading blocks ...
DUDE> Datafile f:\10g\ts_data_02.DBF : reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Done !
DUDE> dump dictionary ;
```

```
DUDE> Dumping OBJ$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows    = 13297
Migrated rows  = 0
Chained rows   = 0
Deleted rows    = 0 (not extracted!)
Skipped rows   = 0 (not extracted!)

DUDE> Done !
DUDE> Dumping TAB$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Done !
DUDE> Dumping COL$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Done !
DUDE> Dumping USER$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Done !
DUDE> create dictionary ;
DUDE> show users ;
DUDE> user# = 28  -  name = TRACESVR
DUDE> user# = 27  -  name = MGMT_VIEW
DUDE> user# = 26  -  name = MGMT_USER
DUDE> user# = 25  -  name = SYSMAN
DUDE> user# = 24  -  name = WM_ADMIN_ROLE
DUDE> user# = 23  -  name = WMSYS
DUDE> user# = 22  -  name = DBSNMP
DUDE> user# = 21  -  name = OEM_MONITOR
DUDE> user# = 20  -  name = HS_ADMIN_ROLE
DUDE> user# = 9  -  name = EXP_FULL_DATABASE
DUDE> user# = 8  -  name = DELETE_CATALOG_ROLE
DUDE> user# = 7  -  name = EXECUTE_CATALOG_ROLE
DUDE> user# = 18  -  name = SCHEDULER_ADMIN
DUDE> user# = 6  -  name = SELECT_CATALOG_ROLE
DUDE> user# = 17  -  name = GLOBAL_AQ_USER_ROLE
DUDE> user# = 5  -  name = SYSTEM
DUDE> user# = 16  -  name = AQ_USER_ROLE
DUDE> user# = 4  -  name = DBA
DUDE> user# = 15  -  name = AQ_ADMINISTRATOR_ROLE
DUDE> user# = 46  -  name = PLUSTRACE
DUDE> user# = 3  -  name = RESOURCE
DUDE> user# = 14  -  name = LOGSTDBY_ADMINISTRATOR
DUDE> user# = 2  -  name = CONNECT
DUDE> user# = 13  -  name = GATHER_SYSTEM_STATISTICS
DUDE> user# = 1  -  name = PUBLIC
DUDE> user# = 12  -  name = RECOVERY_CATALOG_OWNER
DUDE> user# = 0  -  name = SYS
DUDE> user# = 11  -  name = OUTLN
DUDE> user# = 10  -  name = IMP_FULL_DATABASE
DUDE> user# = 40  -  name = SNMPAGENT
DUDE> exit ;
```

14 Addendum A : Valid values for EXPORT_NLS

EXPORT_NLS should reflect the database's characterset.

US7ASCII	DK8BS2000
WE8DEC	S8BS2000
WE8HP	WE8BS2000E
US8PC437	WE8BS2000
WE8EBCDIC37	EE8BS2000
WE8EBCDIC500	CE8BS2000
WE8EBCDIC1140	CL8BS2000
WE8EBCDIC285	WE8BS2000L5
WE8EBCDIC1146	WE8DG
WE8PC850	WE8NCR4970
D7DEC	WE8ROMAN8
F7DEC	EE8MACCE
S7DEC	EE8MACCROATIAN
E7DEC	TR8MACTURKISH
SF7ASCII	IS8MACICELANDIC
NDK7DEC	EL8MACGREEK
I7DEC	IW8MACHEBREW
NL7DEC	US8ICL
CH7DEC	WE8ICL
YUG7ASCII	WE8ISOICLUK
SF7DEC	EE8EBCDIC870C
TR7DEC	EL8EBCDIC875S
IW7IS960	TR8EBCDIC1026S
IN8ISCII	BLT8EBCDIC1112S
WE8EBCDIC1148	IW8EBCDIC424S
WE8PC858	EE8EBCDIC870S
WE8ISO8859P1	CL8EBCDIC1025S
EE8ISO8859P2	TH8TISEBCDICS
SE8ISO8859P3	AR8EBCDIC420S
NEE8ISO8859P4	CL8EBCDIC1025C
CL8ISO8859P5	CL8EBCDIC1025R
AR8ISO8859P6	EL8EBCDIC875R
EL8ISO8859P7	WE8MACROMAN8
IW8ISO8859P8	WE8MACROMAN8S
WE8ISO8859P9	TH8MACTHAI
NE8ISO8859P10	TH8MACTHAIS
TH8TISASCII	HU8CWI2
TH8TISEBCDIC	EL8PC437S
BN8BSCII	EL8EBCDIC875
VN8VN3	EL8PC737
VN8MSWIN1258	LT8PC772
WE8ISO8859P15	LT8PC774
BLT8ISO8859P13	EL8PC869
CEL8ISO8859P14	EL8PC851
CL8ISOIR111	CDN8PC863
WE8NEXTSTEP	HU8ABMOD
CL8KOI8U	AR8ASMO8X
AR8ASMO708PLUS	AR8NAFITHA711T
AR8EBCDICX	AR8SAKHR707T
AR8XBASIC	AR8MUSSAD768T
EL8DEC	AR8ADOS710T
TR8DEC	AR8ADOS720T
WE8EBCDIC37C	AR8APTEC715T
WE8EBCDIC500C	AR8NAFITHA721T
IW8EBCDIC424	AR8HPARABIC8T
TR8EBCDIC1026	AR8NAFITHA711
WE8EBCDIC871	AR8SAKHR707
WE8EBCDIC284	AR8MUSSAD768
WE8EBCDIC1047	AR8ADOS710
WE8EBCDIC1140C	AR8ADOS720

```
WE8EBCDIC1145
WE8EBCDIC1148C
WE8EBCDIC1047E
WE8EBCDIC924
EEC8EUROASCI
EEC8EUROPA3
LA8PASSPORT
BG8PC437S
EE8PC852
RU8PC866
RU8BESTA
IW8PC1507
RU8PC855
TR8PC857
CL8MACCYRILLIC
CL8MACCYRILLICS
WE8PC860
IS8PC861
EE8MACCES
EE8MACCROATIANS
TR8MACTURKISHS
IS8MACICELANDICS
EL8MACGREEKS
IW8MACHEBREWS
EE8MSWIN1250
CL8MSWIN1251
ET8MSWIN923
BG8MSWIN
EL8MSWIN1253
IW8MSWIN1255
LT8MSWIN921
TR8MSWIN1254
WE8MSWIN1252
BLT8MSWIN1257
D8EBCDIC273
I8EBCDIC280
DK8EBCDIC277
S8EBCDIC278
EE8EBCDIC870
CL8EBCDIC1025
F8EBCDIC297
IW8EBCDIC1086
CL8EBCDIC1025X
D8EBCDIC1141
N8PC865
BLT8CP921
LV8PC1117
LV8PC8LR
BLT8EBCDIC1112
LV8RST104090
CL8KOI8R
BLT8PC775
DK8EBCDIC1142
S8EBCDIC1143
I8EBCDIC1144
F7SIEMENS9780X
E7SIEMENS9780X
S7SIEMENS9780X
DK7SIEMENS9780X
N7SIEMENS9780X
I7SIEMENS9780X
D7SIEMENS9780X
F8EBCDIC1147
WE8GCOS7
EL8GCOS7
US8BS2000
D8BS2000
F8BS2000
AR8APTEC715
AR8MSWIN1256
AR8MSAWIN
AR8NAFITHA721
AR8SAKHR706
AR8ARABICMAC
AR8ARABICMACS
AR8ARABICMACT
LA8ISO6937
WE8DECTST
JA16VMS
JA16EUC
JA16EUCYEN
JA16SJIS
JA16DBCS
JA16SJISYEN
JA16EBCDIC930
JA16MACSJIS
JA16EUCTILDE
JA16SJISTILDE
KO16KSC5601
KO16DBCS
KO16KSCCS
KO16MSWIN949
ZHS16CGB231280
ZHS16MACCGB231280
ZHS16GBK
ZHS16DBCS
ZHS32GB18030
ZHT32EUC
ZHT32SOPS
ZHT16DBT
ZHT32TRIS
ZHT16DBCS
ZHT16BIG5
ZHT16CCDC
ZHT16MSWIN950
ZHT16HKSCS
AL24UTFFSS
UTF8
UTFE
AL32UTF8
ZHT32EUCTST
WE16DECTST2
WE16DECTST
KO16TSTSET
JA16TSTSET2
JA16TSTSET
US16TSTFIXED
JA16EUCFIXED
JA16SJISFIXED
JA16DBCSFIXED
KO16KSC5601FIXED
KO16DBCSFIXED
ZHS16CGB231280FIXED
ZHS16GBKFIXED
ZHS16DBCSFIXED
ZHT32EUCFIXED
ZHT32TRISFIXED
ZHT16DBCSFIXED
ZHT16BIG5FIXED
AL16UTF16
HZ-GB-2312
ISO2022-KR
ISO2022-CN
ISO2022-JP
```

E8BS2000	

15 Addendum B : Valid values for FLAT_NLS

DUDE uses the OutputStreamWriter classes to produce flatfiles. During instantiation, a character encoding is passed to the constructor. This character encoding is determined by the *FLAT_NLS* parameter in *dude.cfg*. DUDE uses *ISO8859_1* as default value.

Taken from http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html

Sun's Java 2 Software Development Kit, Standard Edition, v. 1.3.1 for all platforms and the Java 2 Runtime Environment, Standard Edition, v. 1.3.1 for Solaris and Linux support all encodings shown on this page. Sun's Java 2 Runtime Environment, Standard Edition, v. 1.3.1 for Windows comes in two different versions: US-only and international. The US-only version only supports the encodings shown in the first table. The international version (which includes the lib\i18n.jar file) supports all encodings shown on this page.

15.1 Basic Encoding Set (contained in rt.jar)

Canonical Name	Description
ASCII	American Standard Code for Information Interchange
Cp1252	Windows Latin-1
ISO8859_1	ISO 8859-1, Latin alphabet No. 1
UnicodeBig	Sixteen-bit Unicode Transformation Format, big-endian byte order, with byte-order mark
UnicodeBigUnmarked	Sixteen-bit Unicode Transformation Format, big-endian byte order
UnicodeLittle	Sixteen-bit Unicode Transformation Format, little-endian byte order, with byte-order mark
UnicodeLittleUnmarked	Sixteen-bit Unicode Transformation Format, little-endian byte order
UTF8	Eight-bit Unicode Transformation Format
UTF-16	Sixteen-bit Unicode Transformation Format, byte order specified by a mandatory initial byte-order mark

15.2 Extended Encoding Set (contained in i18n.jar)

Canonical Name	Description
Big5	Big5, Traditional Chinese

Big5_HKSCS	Big5 with Hong Kong extensions, Traditional Chinese
Cp037	USA, Canada (Bilingual, French), Netherlands, Portugal, Brazil, Australia
Cp273	IBM Austria, Germany
Cp277	IBM Denmark, Norway
Cp278	IBM Finland, Sweden
Cp280	IBM Italy
Cp284	IBM Catalan/Spain, Spanish Latin America
Cp285	IBM United Kingdom, Ireland
Cp297	IBM France
Cp420	IBM Arabic
Cp424	IBM Hebrew
Cp437	MS-DOS United States, Australia, New Zealand, South Africa
Cp500	EBCDIC 500V1
Cp737	PC Greek
Cp775	PC Baltic
Cp838	IBM Thailand extended SBCS
Cp850	MS-DOS Latin-1
Cp852	MS-DOS Latin-2
Cp855	IBM Cyrillic
Cp856	IBM Hebrew
Cp857	IBM Turkish
Cp858	Variant of Cp850 with Euro character
Cp860	MS-DOS Portuguese
Cp861	MS-DOS Icelandic
Cp862	PC Hebrew
Cp863	MS-DOS Canadian French
Cp864	PC Arabic
Cp865	MS-DOS Nordic
Cp866	MS-DOS Russian
Cp868	MS-DOS Pakistan
Cp869	IBM Modern Greek
Cp870	IBM Multilingual Latin-2
Cp871	IBM Iceland

Cp874	IBM Thai
Cp875	IBM Greek
Cp918	IBM Pakistan (Urdu)
Cp921	IBM Latvia, Lithuania (AIX, DOS)
Cp922	IBM Estonia (AIX, DOS)
Cp930	Japanese Katakana-Kanji mixed with 4370 UDC, superset of 5026
Cp933	Korean Mixed with 1880 UDC, superset of 5029
Cp935	Simplified Chinese Host mixed with 1880 UDC, superset of 5031
Cp937	Traditional Chinese Host miexed with 6204 UDC, superset of 5033
Cp939	Japanese Latin Kanji mixed with 4370 UDC, superset of 5035
Cp942	IBM OS/2 Japanese, superset of Cp932
Cp942C	Variant of Cp942
Cp943	IBM OS/2 Japanese, superset of Cp932 and Shift-JIS
Cp943C	Variant of Cp943
Cp948	OS/2 Chinese (Taiwan) superset of 938
Cp949	PC Korean
Cp949C	Variant of Cp949
Cp950	PC Chinese (Hong Kong, Taiwan)
Cp964	AIX Chinese (Taiwan)
Cp970	AIX Korean
Cp1006	IBM AIX Pakistan (Urdu)
Cp1025	IBM Multilingual Cyrillic: Bulgaria, Bosnia, Herzegovinia, Macedonia (FYR)
Cp1026	IBM Latin-5, Turkey
Cp1046	IBM Arabic - Windows
Cp1097	IBM Iran (Farsi)/Persian
Cp1098	IBM Iran (Farsi)/Persian (PC)
Cp1112	IBM Latvia, Lithuania
Cp1122	IBM Estonia
Cp1123	IBM Ukraine
Cp1124	IBM AIX Ukraine
Cp1140	Variant of Cp037 with Euro character

Cp1141	Variant of Cp273 with Euro character
Cp1142	Variant of Cp277 with Euro character
Cp1143	Variant of Cp278 with Euro character
Cp1144	Variant of Cp280 with Euro character
Cp1145	Variant of Cp284 with Euro character
Cp1146	Variant of Cp285 with Euro character
Cp1147	Variant of Cp297 with Euro character
Cp1148	Variant of Cp500 with Euro character
Cp1149	Variant of Cp871 with Euro character
Cp1250	Windows Eastern European
Cp1251	Windows Cyrillic
Cp1253	Windows Greek
Cp1254	Windows Turkish
Cp1255	Windows Hebrew
Cp1256	Windows Arabic
Cp1257	Windows Baltic
Cp1258	Windows Vietnamese
Cp1381	IBM OS/2, DOS People's Republic of China (PRC)
Cp1383	IBM AIX People's Republic of China (PRC)
Cp33722	IBM-eucJP - Japanese (superset of 5050)
EUC_CN	GB2312, EUC encoding, Simplified Chinese
EUC_JP	JIS X 0201, 0208, 0212, EUC encoding, Japanese
EUC_JP_LINUX	JIS X 0201, 0208, EUC encoding, Japanese
EUC_KR	KS C 5601, EUC encoding, Korean
EUC_TW	CNS11643 (Plane 1-3), EUC encoding, Traditional Chinese
GBK	GBK, Simplified Chinese
ISO2022CN	ISO 2022 CN, Chinese (conversion to Unicode only)
ISO2022CN_CNS	CNS 11643 in ISO 2022 CN form, Traditional Chinese (conversion from Unicode only)
ISO2022CN_GB	GB 2312 in ISO 2022 CN form, Simplified Chinese (conversion from Unicode only)
ISO2022JP	JIS X 0201, 0208 in ISO 2022 form, Japanese
ISO2022KR	ISO 2022 KR, Korean
ISO8859_2	ISO 8859-2, Latin alphabet No. 2

ISO8859_3	ISO 8859-3, Latin alphabet No. 3
ISO8859_4	ISO 8859-4, Latin alphabet No. 4
ISO8859_5	ISO 8859-5, Latin/Cyrillic alphabet
ISO8859_6	ISO 8859-6, Latin/Arabic alphabet
ISO8859_7	ISO 8859-7, Latin/Greek alphabet
ISO8859_8	ISO 8859-8, Latin/Hebrew alphabet
ISO8859_9	ISO 8859-9, Latin alphabet No. 5
ISO8859_13	ISO 8859-13, Latin alphabet No. 7
ISO8859_15_FDIS	ISO 8859-15, Latin alphabet No. 9
JIS0201	JIS X 0201, Japanese
JIS0208	JIS X 0208, Japanese
JIS0212	JIS X 0212, Japanese
JISAutoDetect	Detects and converts from Shift-JIS, EUC-JP, ISO 2022 JP (conversion to Unicode only)
Johab	Johab, Korean
KOI8_R	KOI8-R, Russian
MS874	Windows Thai
MS932	Windows Japanese
MS936	Windows Simplified Chinese
MS949	Windows Korean
MS950	Windows Traditional Chinese
MacArabic	Macintosh Arabic
MacCentralEurope	Macintosh Latin-2
MacCroatian	Macintosh Croatian
MacCyrillic	Macintosh Cyrillic
MacDingbat	Macintosh Dingbat
MacGreek	Macintosh Greek
MacHebrew	Macintosh Hebrew
MacIceland	Macintosh Iceland
MacRoman	Macintosh Roman
MacRomania	Macintosh Romania
MacSymbol	Macintosh Symbol
MacThai	Macintosh Thai
MacTurkish	Macintosh Turkish

MacUkraine	Macintosh Ukraine
SJIS	Shift-JIS, Japanese
TIS620	TIS620, Thai

16 Addendum C : Quick start – SYSTEM available

16.1 Generate a probe file

16.1.1 Generate dude_probe.cfg

This file should include your company details and datafiles you want to extract data from.

Example :

```
USER = "Kurt Van Meerbeeck"
EMAIL = "dude@ora600.org"
COMPANY = "ORA600"
TEL = "+32-495-xxxxxxx"
FAX = "+32-15-xxxxxxx"
OUTPUT_DIR = "c:\\dude_probe"



TABLESPACE "SYSTEM"
{
        DATAFILE="C:\\ORACLE\\ORADATA\\KVMB\\SYSTEM01.DBF"
}

TABLESPACE "EXAMPLE"
{
        DATAFILE="C:\\ORACLE\\ORADATA\\KVMB\\EXAMPLE01.DBF"
}


TABLESPACE "USERS"
{
        DATAFILE="C:\\ORACLE\\ORADATA\\KVMB\\USERS01.DBF"
}
```

16.1.2 Run DUDE_PROBE.jar

Example :

```
C:\oracle_stuff\DUDE_PROBE>java -jar DUDE_PROBE.jar

Probe for DUDE : 1.0 - very limited

NOT FOR DISTRIBUTION 2001 - 2005 Kurt Van Meerbeeck
dude@ora600.org - www.ora600.org - www.miracleas.dk

Roads ? Where we're going we don't need roads ...

Initialising ... : done !

PERSONAL COPY of Kurt Van Meerbeeck
EMAIL = dude@ora600.org
COMPANY = ORA600
TEL = +32-495-xxxxxxx
FAX = +32-15-xxxxxxx
PLATFORM = Windows XP x86
```

```
HOSTNAME = KOERT
IP ADDRESS = 192.168.123.3
VALID1 = 1166397271000
VALID2 = 1167174871000
TABLESPACES = {SYSTEM=TABLESPACE NAME = SYSTEM
 FILENAME = C:\ORACLE\ORADATA\KVMB\SYSTEM01.DBF , SIZE = 304095232
, USERS=TABLESPACE NAME = USERS
 FILENAME = C:\ORACLE\ORADATA\KVMB\USERS01.DBF , SIZE = 191373312
, EXAMPLE=TABLESPACE NAME = EXAMPLE
 FILENAME = C:\ORACLE\ORADATA\KVMB\EXAMPLE01.DBF , SIZE = 110108672
}
```

This will generate a probe.out file – send it to us.

16.2 Generate dude.cfg file

Place this file in the same the directory as the dude.class file.
Example :

```
// sizing parameters
BUFFERCACHE = "1000"
BLOCKSIZE = "8192"

// location parameters
SCAN_DIR = "c:\\oracle_stuff\\dude_scan"
LOG_DIR = "c:\\oracle_stuff\\dude_log"
DUMP_DIR = "c:\\oracle_stuff\\dude_dump"
PLSQL_DIR = "c:\\oracle_stuff\\dude_plsql"
SQLLDR_DIR = "c:\\oracle_stuff\\dude_ctl"
DDL_DIR = "c:\\oracle_stuff\\dude_ddl"
SCRIPT_DIR = "c:\\oracle_stuff\\dude_scan"
LOB_DIR = "c:\\oracle_stuff\\dude_lobs"
DICTIONARY_DIR = "c:\\oracle_stuff\\dude_dic"

// logfile name
LOG_FILE = "dude.log"

// cross-platform unloading
MAP_LOB_DIR="/lobs"
MAP_DUMP_DIR="/dump"
MAP_SQLLDR_DIR="/sqlldr"
MAP_FILESEP="/"

// replace dollar signs with underscore
PRETTY_FILE_NAMES="true"

// auto commands at startup
AUTO_FILENUMBER="true"
AUTO_DICTIONARY="true"
AUTO_BLOCK_CHECK="true"

// platform
PLATFORM="INTEL"  // INTEL/AIX/SOLARIS/HPUX/TRU64/VMS

PARTITIONING="true" // set to "true" if the database contains partitioned tables
ENABLE_LOBS="true" // set to "true" if the database contains LOB columns

EXPORT_MODE="false" // Export Mode (false = flatfile, true = Oracle 805 DMP
file)
// the following parameters are evaluated when EXPORT_MODE="true"
EXPORT_FILE_SIZE="1048576000"
RECORDLENGTH="65536" /// "65536"
```

```
EXPORT_GZIP="false"
GZIP_BUFFER = "5242880"
EXPORT_NLS="WE8ISO8859P1"
EXPORT_NCHAR_NLS="UTF8"
// EOF EXPORT_MODE="true" parameters

// sqlloader flatfile specific
GEN_SQLLDR_CTL="true"
GEN_SQLLDR_STR_MODE="true"  // Generate recordset separator clause in SQLLDR
controlfile (this is only supported in sqlldr 8.1.6 and higher!)
                            // Set to "false" if loading into Oracle 8.0.x
RECORD_SEP="|||\n"   // RECORD SEPERATOR => IMPORTANT : if
GEN_SQLLDR_STR_MODE=FALSE then you must set to "\n" for UNIX and "\r\n" for
WINDOWS
FIELD_SEP="44"
//FIELD_ENCL="34" // "255"
FIELD_ENCL="34"
NULL_FIELD=""
FLAT_NLS="ISO8859_1"

// generate table DDL scripts
GEN_TABLE_DDL="true"

TABLESPACE "SYSTEM"
{
        DATAFILE="C:\\ORACLE\\ORADATA\\KVMB\\SYSTEM01.DBF"
}

TABLESPACE "EXAMPLE"
{
        DATAFILE="C:\\ORACLE\\ORADATA\\KVMB\\EXAMPLE01.DBF"
        ASSM="true"
}


TABLESPACE "USERS"
{
        DATAFILE="C:\\ORACLE\\ORADATA\\KVMB\\USERS01.DBF"
        ASSM="true"
}
```

16.3 Startup DUDE

```
C:\dude_demo>java dude
License information :
User : Kurt Van Meerbeeck
Email : dude@ora600.org
Company : ORA600
Phone : +32-495-xxxxxxx
Fax : +32-15-xxxxxxx
Host : KOERT (192.168.123.3)
Platform : Windows XP x86
Valid for 7 days.


Current size logfile = 970525 bytes

jDUL/DUDE : 2.5.0

NOT FOR DISTRIBUTION 2001 - 2007 Kurt Van Meerbeeck - ORA600 - dude@ora600.org
FOR EDUCATIONAL USE ONLY !
PERSONAL COPY of Kurt Van Meerbeeck / ORA600 / dude@ora600.org
```

```
Roads ? Where we're going we don't need roads ...

DUDE> Initialising ...
DUDE> Init : creating filenumber map ...
DUDE> Scanning tablespace USERS : BLOCKSIZE = 8192
DUDE> File : C:\ORACLE\ORADATA\KVMB\USERS01.DBF resolves to number : 6
DUDE> Scanning tablespace EXAMPLE : BLOCKSIZE = 8192
DUDE> File : C:\ORACLE\ORADATA\KVMB\EXAMPLE01.DBF resolves to number : 3
DUDE> Scanning tablespace SYSTEM : BLOCKSIZE = 8192
DUDE> File : C:\ORACLE\ORADATA\KVMB\SYSTEM01.DBF resolves to number : 1
DUDE> Init : creating dictionary ...
DUDE> Error opening file : c:\oracle_stuff\dude_dic\user.dat
DUDE> Error opening file : c:\oracle_stuff\dude_dic\obj.dat
DUDE> Error opening file : c:\oracle_stuff\dude_dic\col.dat
DUDE> Error opening file : c:\oracle_stuff\dude_dic\tab.dat
DUDE> Error opening file : c:\oracle_stuff\dude_dic\tabpart.dat
DUDE> Error opening file : c:\oracle_stuff\dude_dic\tabsubpart.dat
DUDE> Error opening file : c:\oracle_stuff\dude_dic\lob.dat
DUDE> Init : checking datafile blocksizes ...
DUDE> DATAFILE = C:\ORACLE\ORADATA\KVMB\USERS01.DBF OK : BLOCKSIZE = 8192 equals
 internal blocksize !
DUDE> DATAFILE = C:\ORACLE\ORADATA\KVMB\EXAMPLE01.DBF OK : BLOCKSIZE = 8192 equa
ls internal blocksize !
DUDE> DATAFILE = C:\ORACLE\ORADATA\KVMB\SYSTEM01.DBF OK : BLOCKSIZE = 8192 equal
s internal blocksize !
DUDE> Init done.
DUDE>
```

16.4 Create blockmaps for all tablespaces

```
DUDE> create blockmap for tablespace SYSTEM ;
DUDE> ID := 0 BLOCKMAPPER for TABLESPACE NAME = SYSTEM
OFFSET = 0
ASSM = false
BIGFILE = false
BLOCKSIZE = 8192
NUMBER = 1 FILENAME = C:\ORACLE\ORADATA\KVMB\SYSTEM01.DBF IBS = 8192


DUDE> Datafile C:\ORACLE\ORADATA\KVMB\SYSTEM01.DBF : start reading blocks ...
DUDE> Datafile C:\ORACLE\ORADATA\KVMB\SYSTEM01.DBF : reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Please wait for chunk stream to finish ...
DUDE> Done !
DUDE> create blockmap for tablespace EXAMPLE ;
DUDE> ID := 0 BLOCKMAPPER for TABLESPACE NAME = EXAMPLE
OFFSET = 0
ASSM = true
BIGFILE = false
BLOCKSIZE = 8192
NUMBER = 3 FILENAME = C:\ORACLE\ORADATA\KVMB\EXAMPLE01.DBF IBS = 8192


DUDE> Datafile C:\ORACLE\ORADATA\KVMB\EXAMPLE01.DBF : start reading blocks ...
DUDE> Datafile C:\ORACLE\ORADATA\KVMB\EXAMPLE01.DBF : reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Please wait for chunk stream to finish ...
DUDE> Done !
DUDE> create blockmap for tablespace USERS ;
DUDE> ID := 0 BLOCKMAPPER for TABLESPACE NAME = USERS
OFFSET = 0
ASSM = true
BIGFILE = false
BLOCKSIZE = 8192
```

```
NUMBER = 6 FILENAME = C:\ORACLE\ORADATA\KVMB\USERS01.DBF IBS = 8192


DUDE> Datafile C:\ORACLE\ORADATA\KVMB\USERS01.DBF : start reading blocks ...
DUDE> Datafile C:\ORACLE\ORADATA\KVMB\USERS01.DBF : reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Please wait for chunk stream to finish ...
DUDE> Done !
DUDE>
```

16.5 Dump the dictionary

```
DUDE> dump dictionary ;
DUDE> Dumping OBJ$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows    = 25351
Migrated rows  = 0
Chained rows   = 0
Deleted rows    = 56 (not extracted!)
Skipped rows   = 0 (not extracted!)

DUDE> Done !
DUDE> Dumping TAB$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Done !
DUDE> Dumping TABPART$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows    = 55
Migrated rows  = 0
Chained rows   = 0
Deleted rows    = 23 (not extracted!)
Skipped rows   = 0 (not extracted!)

DUDE> Done !
DUDE> Dumping TABSUBPART$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows    = 24
Migrated rows  = 0
Chained rows   = 0
Deleted rows    = 0 (not extracted!)
Skipped rows   = 0 (not extracted!)

DUDE> Done !
DUDE> Dumping INDPART$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows    = 56
Migrated rows  = 0
Chained rows   = 0
Deleted rows    = 80 (not extracted!)
Skipped rows   = 0 (not extracted!)

DUDE> Done !
DUDE> Dumping IND$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
```

```
DUDE> Done !
DUDE> Dumping COL$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Done !
DUDE> Dumping USER$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Done !
DUDE> Dumping LOB$...
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Done !
DUDE>
```

16.6 Create the dictionary in DUDE

```
DUDE> create dictionary ;
DUDE>
```

16.7 Test the dictionary by list the users

```
DUDE> show users ;
DUDE> user# = 29  -  name = JAVA_DEPLOY
DUDE> user# = 28  -  name = JAVA_ADMIN
DUDE> user# = 27  -  name = EJBCLIENT
DUDE> user# = 26  -  name = JAVADEBUGPRIV
DUDE> user# = 25  -  name = JAVASYSPRIV
DUDE> user# = 24  -  name = JAVAIDPRIV
DUDE> user# = 23  -  name = JAVAUSERPRIV
DUDE> user# = 22  -  name = WM_ADMIN_ROLE
DUDE> user# = 21  -  name = WMSYS
DUDE> user# = 20  -  name = HS_ADMIN_ROLE
DUDE> user# = 9  -  name = EXP_FULL_DATABASE
DUDE> user# = 8  -  name = DELETE_CATALOG_ROLE
DUDE> user# = 19  -  name = DBSNMP
DUDE> user# = 7  -  name = EXECUTE_CATALOG_ROLE
DUDE> user# = 18  -  name = OEM_MONITOR
DUDE> user# = 6  -  name = SELECT_CATALOG_ROLE
DUDE> user# = 17  -  name = GLOBAL_AQ_USER_ROLE
DUDE> user# = 5  -  name = SYSTEM
DUDE> user# = 16  -  name = AQ_USER_ROLE
DUDE> user# = 4  -  name = DBA
DUDE> user# = 15  -  name = AQ_ADMINISTRATOR_ROLE
DUDE> user# = 47  -  name = INSIGHT
DUDE> user# = 46  -  name = K
DUDE> user# = 3  -  name = RESOURCE
DUDE> user# = 14  -  name = LOGSTDBY_ADMINISTRATOR
DUDE> user# = 45  -  name = KVMB
DUDE> user# = 2  -  name = CONNECT
DUDE> user# = 13  -  name = GATHER_SYSTEM_STATISTICS
DUDE> user# = 44  -  name = SH
DUDE> user# = 1  -  name = PUBLIC
DUDE> user# = 12  -  name = RECOVERY_CATALOG_OWNER
DUDE> user# = 43  -  name = PM
DUDE> user# = 0  -  name = SYS
DUDE> user# = 11  -  name = OUTLN
DUDE> user# = 42  -  name = OE
DUDE> user# = 10  -  name = IMP_FULL_DATABASE
```

```
DUDE> user# = 41  -  name = HR
DUDE> user# = 36  -  name = SALES_HISTORY_ROLE
DUDE> user# = 33  -  name = AUTHENTICATEDUSER
DUDE> user# = 32  -  name = XDBADMIN
DUDE> user# = 31  -  name = ANONYMOUS
DUDE> user# = 30  -  name = XDB
DUDE>
```

16.8 Dump the data

```
DUDE>dump tablespace EXAMPLE ;
DUDE>dump tablespace USERS ;

Or

DUDE>dump user SH ;

Or

DUDE>dump table SH SALES ;

Or

DUDE>dump objected 25575 ;
```

17 Addendum D : Quick start – SYSTEM not available

Remember – if you've lost the SYSTEM tablespace, you've lost the DICTIONARY. So there's no way to resolve an objectid to an owner, table name, column name & datatypes !
It's therefore recommended that you dump data to flatfiles so that data can be visually identified. This requires thorough knowledge of your data and table layouts.

17.1 Generate a probe file

17.1.1 Generate dude_probe.cfg

This file should include your company details and datafiles you want to extract data from.

Example :

```
USER = "Kurt Van Meerbeeck"
EMAIL = "dude@ora600.org"
COMPANY = "ORA600"
TEL = "+32-495-xxxxxxx"
FAX = "+32-15-xxxxxxx"
OUTPUT_DIR = "c:\\dude_probe"



TABLESPACE "EXAMPLE"
{
        DATAFILE="C:\\DUDE\\DBF\\EXAMPLE01.DBF"
}
```

17.1.2 Run DUDE_PROBE.jar

Example :

```
C:\oracle_stuff\DUDE_PROBE>java -jar DUDE_PROBE.jar

Probe for DUDE : 1.0 - very limited

NOT FOR DISTRIBUTION 2001 - 2005 Kurt Van Meerbeeck
dude@ora600.org - www.ora600.org - www.miracleas.dk

Roads ? Where we're going we don't need roads ...

Initialising ... : done !

PERSONAL COPY of Kurt Van Meerbeeck
EMAIL = dude@ora600.org
COMPANY = ORA600
TEL = +32-495-xxxxxxx
FAX = +32-15-xxxxxxx
PLATFORM = Windows XP x86
HOSTNAME = KOERT
```

```
IP ADDRESS = 192.168.123.3
VALID1 = 1166397271000
VALID2 = 1167174871000
TABLESPACES = { EXAMPLE=TABLESPACE NAME = EXAMPLE
 FILENAME = C:\ORACLE\ORADATA\KVMB\EXAMPLE01.DBF , SIZE = 110108672
}
```

This will generate a probe.out file – send it to us.

17.2 Generate dude.cfg file

Place this file in the same the directory as the dude.class file.
Make sure to specify SAMPLE_SIZE and SAMPLE_LIMIT in the tablespace clause !

Also for convenience – set SCAN_DIR equal to SCRIPT_DIR. This way you can run generated *.dde files, containing the dump command, directly using the **start** command.

Example :

```
// sizing parameters
BUFFERCACHE = "1000"
BLOCKSIZE = "8192"

// location parameters
SCAN_DIR = "c:\\oracle_stuff\\dude_scan"
LOG_DIR = "c:\\oracle_stuff\\dude_log"
DUMP_DIR = "c:\\oracle_stuff\\dude_dump"
PLSQL_DIR = "c:\\oracle_stuff\\dude_plsql"
SQLLDR_DIR = "c:\\oracle_stuff\\dude_ctl"
DDL_DIR = "c:\\oracle_stuff\\dude_ddl"
SCRIPT_DIR = "c:\\oracle_stuff\\dude_scan"
LOB_DIR = "c:\\oracle_stuff\\dude_lobs"
DICTIONARY_DIR = "c:\\oracle_stuff\\dude_dic"

// logfile name
LOG_FILE = "dude.log"

// cross-platform unloading
MAP_LOB_DIR="/lobs"
MAP_DUMP_DIR="/dump"
MAP_SQLLDR_DIR="/sqlldr"
MAP_FILESEP="/"

// replace dollar signs with underscore
PRETTY_FILE_NAMES="true"

// auto commands at startup
AUTO_FILENUMBER="true"
AUTO_DICTIONARY="true"
AUTO_BLOCK_CHECK="true"

// platform
PLATFORM="INTEL"  // INTEL/AIX/SOLARIS/HPUX/TRU64/VMS

PARTITIONING="true" // set to "true" if the database contains partitioned tables
ENABLE_LOBS="true" // set to "true" if the database contains LOB columns
```

```
EXPORT_MODE="false" // Export Mode (false = flatfile, true = Oracle 805 DMP
file)
// the following parameters are evaluated when EXPORT_MODE="true"
EXPORT_FILE_SIZE="1048576000"
RECORDLENGTH="65536" /// "65536"
EXPORT_GZIP="false"
GZIP_BUFFER = "5242880"
EXPORT_NLS="WE8ISO8859P1"
EXPORT_NCHAR_NLS="UTF8"
// EOF EXPORT_MODE="true" parameters

// sqlloader flatfile specific
GEN_SQLLDR_CTL="true"
GEN_SQLLDR_STR_MODE="true"  // Generate recordset separator clause in SQLLDR
controlfile (this is only supported in sqlldr 8.1.6 and higher!)
                            // Set to "false" if loading into Oracle 8.0.x
RECORD_SEP="|||\n"   // RECORD SEPERATOR => IMPORTANT : if
GEN_SQLLDR_STR_MODE=FALSE then you must set to "\n" for UNIX and "\r\n" for
WINDOWS
FIELD_SEP="44"
//FIELD_ENCL="34" // "255"
FIELD_ENCL="34"
NULL_FIELD=""
FLAT_NLS="ISO8859_1"

// generate table DDL scripts
GEN_TABLE_DDL="true"

TABLESPACE "EXAMPLE"
{
        DATAFILE="C:\\DUDE\\DBF\\EXAMPLE01.DBF"
        BLOCKSIZE="8192"
        ASSM="true"
        BIGFILE="false"
        SAMPLE_LIMIT="100"
        SAMPLE_SIZE="10000"
}
```

17.3 Startup DUDE

```
C:\dude_demo>java dude
License information :
User : Kurt Van Meerbeeck
Email : dude@ora600.org
Company : ORA600
Phone : +32-495-xxxxxxx
Fax : +32-15-xxxxxxx
Host : KOERT (192.168.123.3)
Platform : Windows XP x86
Valid for 7 days.

Current size logfile = 35051 bytes

jDUL/DUDE : 2.5.7

NOT FOR DISTRIBUTION 2001 - 2007 Kurt Van Meerbeeck - ORA600 - dude@ora600.org
FOR EDUCATIONAL USE ONLY !
PERSONAL COPY of Thomas Presslie / Miracle LTD / tpr@miracleltd.com

Roads ? Where we're going we don't need roads ...

DUDE> Initialising ...
```

```
DUDE> Init : creating filenumber map ...
DUDE> Scanning tablespace EXAMPLE : BLOCKSIZE = 8192
DUDE> File : C:\DUDE\DBF\EXAMPLE01.DBF resolves to number : 3
DUDE> Init : checking datafile blocksizes ...
DUDE> DATAFILE = C:\DUDE\DBF\EXAMPLE01.DBF OK : BLOCKSIZE = 8192 equals internal
 blocksize !
DUDE> Init done.
```

17.4 Create blockmaps for all tablespaces

```
DUDE> create blockmap for tablespace EXAMPLE ;
DUDE> ID := 0 BLOCKMAPPER for TABLESPACE NAME = EXAMPLE
OFFSET = 0
ASSM = true
BIGFILE = false
BLOCKSIZE = 8192
NUMBER = 3 FILENAME = C:\DUDE\DBF\EXAMPLE01.DBF IBS = 8192


DUDE> Datafile C:\DUDE\DBF\EXAMPLE01.DBF : start reading blocks ...
DUDE> Datafile C:\DUDE\DBF\EXAMPLE01.DBF : reading blocks done !
DUDE> Block type 0 : 1193 blocks -> 9544Kb -> 93%
DUDE> Block type 6 : 38 blocks -> 304Kb -> 3%
DUDE> Block type 11 : 1 blocks -> 8Kb -> 0%
DUDE> Block type 29 : 1 blocks -> 8Kb -> 0%
DUDE> Block type 30 : 6 blocks -> 48Kb -> 0%
DUDE> Block type 32 : 14 blocks -> 112Kb -> 1%
DUDE> Block type 33 : 14 blocks -> 112Kb -> 1%
DUDE> Block type 35 : 14 blocks -> 112Kb -> 1%
DUDE> Total Blocks scanned = 1281
DUDE> Please wait for output stream to finish ...
DUDE> Done !
DUDE>
```

17.5 Scan the tablespace for segments

```
DUDE> scan tablespace EXAMPLE ;
DUDE> Scanning segments ...
DUDE> Scanning OBJECTID : 25281
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
No rowstats available ! Rows sampled : 0

DUDE> Done !
DUDE> Scanning OBJECTID : 25279
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
Rows sampled : 107
COL 0 : LEN = 3 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 21% : NICE NUM = 10
0%
COL 1 : LEN = 11 : NULLS = 0% : POSS. DATE = 22% : PRINTABLE = 100% : NICE NUM =
 27%
COL 2 : LEN = 11 : NULLS = 0% : POSS. DATE = 14% : PRINTABLE = 100% : NICE NUM =
 36%
COL 3 : LEN = 8 : NULLS = 0% : POSS. DATE = 30% : PRINTABLE = 100% : NICE NUM =
26%
COL 4 : LEN = 18 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 100% : NICE NUM =
```

```
100%
COL 5 : LEN = 7 : NULLS = 0% : POSS. DATE = 100% : PRINTABLE = 15% : NICE NUM =
0%
COL 6 : LEN = 10 : NULLS = 0% : POSS. DATE = 6% : PRINTABLE = 100% : NICE NUM =
6%
COL 7 : LEN = 3 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 29% : NICE NUM = 10
0%
COL 8 : LEN = 2 : NULLS = 67% : POSS. DATE = 0% : PRINTABLE = 5% : NICE NUM = 10
0%
COL 9 : LEN = 3 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 9% : NICE NUM = 100
%
COL 10 : LEN = 3 : NULLS = 0% : POSS. DATE = 0% : PRINTABLE = 41% : NICE NUM = 9
9%

DUDE> Done !
DUDE> Scanning OBJECTID : 25280
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
No rowstats available ! Rows sampled : 0

DUDE> Done !
DUDE> Scanning OBJECTID : 25278
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
No rowstats available ! Rows sampled : 0

DUDE> Done !
DUDE> Scanning OBJECTID : 25277
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
Rows sampled : 19
COL 0 : LEN = 10 : NULLS = 0% : POSS. DATE = 15% : PRINTABLE = 100% : NICE NUM =
 31%
COL 1 : LEN = 31 : NULLS = 0% : POSS. DATE = 5% : PRINTABLE = 100% : NICE NUM =
5%
COL 2 : LEN = 3 : NULLS = 0% : POSS. DATE = 5% : PRINTABLE = 33% : NICE NUM = 10
0%
COL 3 : LEN = 3 : NULLS = 0% : POSS. DATE = 5% : PRINTABLE = 29% : NICE NUM = 10
0%

DUDE> Done !
DUDE> Scanning OBJECTID : 25275
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
No rowstats available ! Rows sampled : 0

DUDE> Done !
DUDE> Scanning OBJECTID : 25274
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
Rows sampled : 27
COL 0 : LEN = 3 : NULLS = 0% : POSS. DATE = 3% : PRINTABLE = 22% : NICE NUM = 10
0%
COL 1 : LEN = 20 : NULLS = 0% : POSS. DATE = 11% : PRINTABLE = 100% : NICE NUM =
 29%
COL 2 : LEN = 3 : NULLS = 59% : POSS. DATE = 9% : PRINTABLE = 3% : NICE NUM = 10
0%
COL 3 : LEN = 2 : NULLS = 0% : POSS. DATE = 3% : PRINTABLE = 0% : NICE NUM = 100
```

```
%

DUDE> Done !
DUDE> Scanning OBJECTID : 25272
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
No rowstats available ! Rows sampled : 0

DUDE> Done !
DUDE> Scanning OBJECTID : 25271
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
Rows sampled : 23
COL 0 : LEN = 2 : NULLS = 0% : POSS. DATE = 4% : PRINTABLE = 4% : NICE NUM = 100
%
COL 1 : LEN = 40 : NULLS = 0% : POSS. DATE = 4% : PRINTABLE = 100% : NICE NUM =
65%
COL 2 : LEN = 11 : NULLS = 4% : POSS. DATE = 18% : PRINTABLE = 100% : NICE NUM =
 40%
COL 3 : LEN = 19 : NULLS = 0% : POSS. DATE = 21% : PRINTABLE = 100% : NICE NUM =
 13%
COL 4 : LEN = 17 : NULLS = 26% : POSS. DATE = 17% : PRINTABLE = 99% : NICE NUM =
 17%
COL 5 : LEN = 2 : NULLS = 0% : POSS. DATE = 4% : PRINTABLE = 100% : NICE NUM = 3
4%

DUDE> Done !
DUDE> Scanning OBJECTID : 25268
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
No rowstats available ! Rows sampled : 0

DUDE> Done !
DUDE> Scanning OBJECTID : 25270
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
No rowstats available ! Rows sampled : 0

DUDE> Done !
DUDE> Scanning OBJECTID : 25267
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
Rows sampled : 4
COL 0 : LEN = 2 : NULLS = 0% : POSS. DATE = 25% : PRINTABLE = 0% : NICE NUM = 10
0%
COL 1 : LEN = 22 : NULLS = 0% : POSS. DATE = 25% : PRINTABLE = 100% : NICE NUM =
 75%

DUDE> Done !
DUDE> Scanning OBJECTID : 25284
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
No rowstats available ! Rows sampled : 0

DUDE> Done !
DUDE> Scanning OBJECTID : 25283
```

```
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE> Number of rows skipped : 0
DUDE>
Rows sampled : 10
COL 0 : LEN = 3 : NULLS = 0% : POSS. DATE = 10% : PRINTABLE = 7% : NICE NUM = 10
0%
COL 1 : LEN = 7 : NULLS = 0% : POSS. DATE = 100% : PRINTABLE = 14% : NICE NUM =
0%
COL 2 : LEN = 7 : NULLS = 0% : POSS. DATE = 100% : PRINTABLE = 14% : NICE NUM =
0%
COL 3 : LEN = 10 : NULLS = 0% : POSS. DATE = 20% : PRINTABLE = 100% : NICE NUM =
 40%
COL 4 : LEN = 3 : NULLS = 0% : POSS. DATE = 10% : PRINTABLE = 31% : NICE NUM = 1
00%

DUDE> Done !
DUDE>
```

17.6 Dump the scanned tables/tablespaces

After the tablespace scan, you'll find a DDE file for each segment in the SCAN_DIR directory. These files contain the dump commands for unloading the segment's data.
You can either copy & paste the content of those file into DUDE or you can use the *START* command to run the DDE files in DUDE. (if you have set *SCRIPT_DIR = SCAN_DIR*).
To dump the complete tablespace, use the *DUMP SCANNED TABLESPACE* command – this will run each DDE file in DUDE.

```
Directory of C:\DUDE\scan

09/06/2007  17:19    <DIR>          .
09/06/2007  17:19    <DIR>          ..
09/06/2007  17:19                61 SCAN_25267.dde
09/06/2007  17:19                32 SCAN_25268.dde
09/06/2007  17:19                32 SCAN_25270.dde
09/06/2007  17:19               113 SCAN_25271.dde
09/06/2007  17:19                32 SCAN_25272.dde
09/06/2007  17:19                91 SCAN_25274.dde
09/06/2007  17:19                32 SCAN_25275.dde
09/06/2007  17:19                89 SCAN_25277.dde
09/06/2007  17:19                32 SCAN_25278.dde
09/06/2007  17:19               187 SCAN_25279.dde
09/06/2007  17:19                32 SCAN_25280.dde
09/06/2007  17:19                32 SCAN_25281.dde
09/06/2007  17:19               102 SCAN_25283.dde
09/06/2007  17:19                32 SCAN_25284.dde
              14 File(s)            899 bytes
               2 Dir(s)   8.252.796.928 bytes free
```

If you detect a wrong datatype in the DDE file, you can simple edit it and run it again.

```
DUDE>dump scanned tablespace EXAMPLE ;

Or
```

```
DUDE>start SCAN_25283.dde ;
```

```
DUDE> start SCAN_25283.dde ;
DUDE> dump SCANNED OBJECTID 25283 (  COL0 NUMBER ,  COL1 DATE ,  COL2 DATE ,  CO
L3 CHAR ,  COL4 NUMBER ) ;
DUDE> Dumping objectid 25283
DUDE> Reading blocks done !
DUDE> Please wait for output stream to finish ...
DUDE>
Normal rows     = 10
Migrated rows   = 0
Chained rows    = 0
Deleted rows     = 0 (not extracted!)
Skipped rows    = 0 (not extracted!)

DUDE> Done !
DUDE> Dumping objectid 25283 Done !
```

17.7 Check dumped data

Check the generate *.dat or *.dmp files in *DUMP_DIR*.
Because there's no data dictionary (no SYSTEM !) you'll notice that all files are named like this :
SCANNED_<DATAOBJECTID>.dat
Or reflecting our example :
SCANNED_25575.dat

17.8 Identify the data

Identify the data by opening the *SCANNED_<DATAOBJECTID>.dat* files one by one.
Once you've identified a flatfile, you can :

- Recreate the table in a new database
- Adjust the generated sqlldr controlfile in *SQLLDR_DIR* to reflect the table name :

```
LOAD DATA
INFILE 'c:\dude\dump\SCANNED_25283.dat' "str X'7c7c7c0a'"
BADFILE 'c:\dude\sqlldr\KOERT_OBJ25283_25283_BAD.dat'
INSERT
INTO TABLE "OBJ25283"
FIELDS TERMINATED BY X'2c' ENCLOSED BY X'22'
(
COL0                DECIMAL EXTERNAL,
COL1                DATE "DD-MM-YYYY BC HH24:MI:SS",
COL2                DATE "DD-MM-YYYY BC HH24:MI:SS",
COL3                CHAR,
COL4                DECIMAL EXTERNAL)
```

Change the table name in the sqlldr controlfile :

```
LOAD DATA
INFILE 'c:\dude\dump\SCANNED_25283.dat' "str X'7c7c7c0a'"
BADFILE 'c:\dude\sqlldr\KOERT_OBJ25283_25283_BAD.dat'
INSERT
INTO TABLE "MYTABLE"
FIELDS TERMINATED BY X'2c' ENCLOSED BY X'22'
(
COL0                DECIMAL EXTERNAL,
COL1                DATE "DD-MM-YYYY BC HH24:MI:SS",
COL2                DATE "DD-MM-YYYY BC HH24:MI:SS",
COL3                CHAR,
COL4                DECIMAL EXTERNAL)
```

17.9 Load the data

Using sqlldr :

- sqlldr username/password control=KOERT_OBJ25283_25283.ctl
- check the sqlldr logfile and badfile after the dataload

18 Troubleshooting

18.1 java.lang.OutOfMemoryError exception (1)

This is a typical java exception when a program runs out of heap space. Startup DUDE with a higher maximum heap size :

Example – startup with heapsize of 512Mb
Java –Xmx512m dude

18.2 java.lang.OutOfMemoryError exception (2)

If DUDE still runs out of memory after setting the heapsize to 2Gb (maxium), then most probably you are dumping large LOB columns. If this is the case, you should switch on the parameter *HUGE_LOB_SUPPORT*.
Setting *HUGE_LOB_SUPPORT="true"* in dude.cfg will turn on an on-disk-hashing algorithm for unloading lobs (as opposed to in-memory hashing).

18.3 DUMP DICTIONARY throws lots of errors

There can be many reasons for this of course.

18.3.1 ASCII mode FTP

Many people ftp their datafiles of their server to another machine. Make sure you use binary mode ftp. In case of ASCII mode you see the following symptoms while blockmapping :

```
04-07-2007 13:50 : DUDE> Datafile C:\ORCL\system01.dbf : start reading blocks
...
04-07-2007 13:50 : DUDE> Datafile C:\ORCL\system01.dbf : reading blocks done !
04-07-2007 13:50 : DUDE> Block type 0 : 38230 blocks -> 305840Kb -> 54%
04-07-2007 13:50 : DUDE> Block type 1 : 1474 blocks -> 11792Kb -> 2%
04-07-2007 13:50 : DUDE> Block type 2 : 1709 blocks -> 13672Kb -> 2%
04-07-2007 13:50 : DUDE> Block type 3 : 760 blocks -> 6080Kb -> 1%
04-07-2007 13:50 : DUDE> Block type 4 : 912 blocks -> 7296Kb -> 1%
04-07-2007 13:50 : DUDE> Block type 5 : 369 blocks -> 2952Kb -> 1%
04-07-2007 13:50 : DUDE> Block type 6 : 612 blocks -> 4896Kb -> 1%
04-07-2007 13:50 : DUDE> Block type 7 : 352 blocks -> 2816Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 8 : 242 blocks -> 1936Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 9 : 218 blocks -> 1744Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 10 : 267 blocks -> 2136Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 11 : 147 blocks -> 1176Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 12 : 167 blocks -> 1336Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 13 : 383 blocks -> 3064Kb -> 1%
```

```
04-07-2007 13:50 : DUDE> Block type 14 : 133 blocks -> 1064Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 15 : 109 blocks -> 872Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 16 : 191 blocks -> 1528Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 17 : 159 blocks -> 1272Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 18 : 131 blocks -> 1048Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 19 : 105 blocks -> 840Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 20 : 204 blocks -> 1632Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 21 : 126 blocks -> 1008Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 22 : 124 blocks -> 992Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 23 : 105 blocks -> 840Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 24 : 117 blocks -> 936Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 25 : 154 blocks -> 1232Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 26 : 94 blocks -> 752Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 27 : 96 blocks -> 768Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 28 : 123 blocks -> 984Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 29 : 103 blocks -> 824Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 30 : 215 blocks -> 1720Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 31 : 126 blocks -> 1008Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 32 : 979 blocks -> 7832Kb -> 1%
04-07-2007 13:50 : DUDE> Block type 33 : 84 blocks -> 672Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 34 : 91 blocks -> 728Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 35 : 68 blocks -> 544Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 36 : 101 blocks -> 808Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 37 : 75 blocks -> 600Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 38 : 73 blocks -> 584Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 39 : 99 blocks -> 792Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 40 : 117 blocks -> 936Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 41 : 117 blocks -> 936Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 42 : 146 blocks -> 1168Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 43 : 116 blocks -> 928Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 44 : 248 blocks -> 1984Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 45 : 218 blocks -> 1744Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 46 : 109 blocks -> 872Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 47 : 328 blocks -> 2624Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 48 : 204 blocks -> 1632Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 49 : 132 blocks -> 1056Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 50 : 129 blocks -> 1032Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 51 : 97 blocks -> 776Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 52 : 102 blocks -> 816Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 53 : 101 blocks -> 808Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 54 : 106 blocks -> 848Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 55 : 96 blocks -> 768Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 56 : 112 blocks -> 896Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 57 : 97 blocks -> 776Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 58 : 70 blocks -> 560Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 59 : 90 blocks -> 720Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 60 : 71 blocks -> 568Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 61 : 111 blocks -> 888Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 62 : 59 blocks -> 472Kb -> 0%
04-07-2007 13:50 : DUDE> Block type 63 : 17874 blocks -> 142992Kb -> 25%
04-07-2007 13:50 : DUDE> Total Blocks scanned = 70677
04-07-2007 13:50 : DUDE> Please wait for output stream to finish ...
04-07-2007 13:50 : DUDE> Done !
```

What's wrong with this picture ?
Well, normally you shouldn't have 64 different blocktypes in a datafile. And you should have a reasonable amount of blocktype 0x06 blocks (which is the data you want). So if you see this while blockmapping … you've used ASCII mode ftp on your datafiles.

18.3.2 Wrong PLATFORM parameter

If you're lucky you'll pick this one up while starting up DUDE as the file numbers resolve to something ridicules. For example, the first datafile of SYSTEM has normally file# =1 (unless it's an Oracle7 or migrated Oracle7). If this is not the case then maybe you have your PLATFORM set to a wrong value. Remember, it should be the platform where the database ran on – not where DUDE is currently running on.
Now, if you continue blockmapping and then run the 'dump dictionary' command, a lot of errors will be spawned.

18.4 All tables return with 0 records while dumping

Toggle the ASSM parameter in the tablespace clause. Most probably you defined your tablespace as ASSM="true" but it was not created with "automatic segment space management". (or vice versa)
Some tips :

- ASSM was introduced in 9i – so if you're on 7,8.0 or 8i, always set ASSM="false" (default)
- SYSTEM never uses ASSM
- It's not necessary to drop and recreate the blockmap of a tablespace if you toggle ASSM

18.5 DUDE tries to load a block from a ridicules blocknumber

Typically you'll find the following error message thrown at you :
Error retrieving datablock : file# = 432 offset = 2891
Error retrieving datablock : file# = 444 offset = 2118625
Error retrieving datablock : file# = 269 offset = 3108468

The error is suspicious because in this particular case DUDE tries to load blocks from a broad range of datafiles (432, 444, 269) that do not exist !

Typically this happens when a block is fractured. Chances are that one part of the block is a different version then rest of the block, and thus, an entry in the rowdirectory might point to an invalid row header. Now, if that bogus row header is identified as a chained rowpiece, then DUDE will pick up a bogus block address for the next row piece !

To avoid this – you can do the following things :

- Set FLAG_FRACTURED_BLOCK="true" : in this case, the checksum of the block is calculated and compared to the checksum stored in the block. If the values don't match, the block is fractured or corrupt and a warning is shown. No other action is taken and DUDE will try to unload the complete block.
- Set SKIP_OUT_OF_SYNC="true" : same as *FLAG_FRACTURED_BLOCK*, however, **bogus ROW headers will be SKIPPED** !!!

- Set SKIP_FRACTURED_BLOCK="true" : same as *FLAG_FRACTURED_BLOCK,* however, **the COMPLETE fractured BLOCK is SKIPPED !!!**

It's recommended that if you encounter '*Error retrieving datablock : file#*' errors, you switch on *SKIP_OUT_OF_SYNC* and dump the data to **flatfiles** as they are less error-phrone and can be easily altered in case of errors during loading.

18.6 Combination of all the above problems

You might be using GNU java. Do not use GNU java – download a Sun's java environment from www.javasoft.com.

19 Acknowledgements

I would like to express my gratitude for their support and assistance while developing DUDE :

- Kugendran Naidoo / NRG Consulting, South Africa
- Pete Finnigan / http://www.petefinnigan.com/
- Mogens Nørgaard, Henrik Rasmussen, Michael Möller, Kaj Christensen, Johannes Djernaes, Peter Gram / Miracle AS (www.miracleas.dk)
- Thomas Presslie / Miracle LTD (www.miracleltd.com)
- Daniel Fink / OptimalDBA (www.optimaldba.com)
- Tim Gorman / Evergreen Database Technologies (www.evdbt.com)
- Doug Burns
- Michel Balliere, Els De Bie, Steven Janssens / AXI NV/BV (www.axi.be / www.axi.nl)
- And the boys of *the* coolest DBA team in Belgium : Kris Welleman, Bart Van Den Bergh, John Van Den Eynde, Yves Van Wert, Peter Clerbout, Luc Vlaeminckx
- Fellows of the Oaktable Network / (www.oaktable.net)