Disassembling the Oracle Data Block

A Guide to the bbed Block Browser and Editor

Written by Graham Thornton

September 19th 2005

Contents

Introduction

The following document discusses the Oracle bbed tool. The name bbed is an acronym for Block Browser and EDitor and it is shipped with the database. It is intended for Oracle internal use only and the company never publishes any details about it. It is a very powerful tool but also extremely dangerous since it can change and/or corrupt data blocks of any Oracle database.

If you use this tool, you do so at your own risk. Any modifications made with this tool render the database unsupported by Oracle.

Linking bbed

Before the bbed tool can be used, it must be linked. The object code is shipped and installed with Oracle databases on UNIX and Linux, but it is not linked by the installer. To link it, shift to the rdbms/install directory and issue the following command:

```
[oracle@RDBALINUX03 oracle]$ cd $ORACLE_HOME/rdbms/lib
[oracle@RDBALINUX03 lib]$ make -f ins_rdbms.mk $ORACLE_HOME/rdbms/lib/bbed

Linking BBED utility (bbed)
rm -f /home/oracle/Ora92/rdbms/lib/bbed
gcc -o /home/oracle/Ora92/rdbms/lib/bbed -L/home/oracle/Ora92/rdbms/lib/ -
L/home/oracle/Ora92/lib/ -L/home/oracle/Ora92/lib/stubs/  /home/oracle/Ora92/lib/s0main.o
/home/oracle/Ora92/rdbms/lib/ssbbded.o /home/oracle/Ora92/rdbms/lib/sbbdpt.o `cat
<output removed to aid clarity>
```

```
-lcore9 -lunls9 -lnls9   `cat /home/oracle/Ora92/lib/sysliblist` -Wl,-
rpath,/home/oracle/Ora92/lib:/lib:/usr/lib -lm    `cat /home/oracle/Ora92/lib/sysliblist`
-ldl -lm
```

You can now check to see if the bbed tool was linked by using the ls command.

```
[oracle@RDBALINUX03 lib]$ ls -al $ORACLE_HOME/rdbms/lib/bbed
-rwxrwxr-x    1 oracle   oracle     321165 Sep 19 09:10 /home/oracle/Ora92/rdbms/lib/bbed
[oracle@RDBALINUX03 lib]$
```

Starting bbed

By default, the bbed tool is linked in the rdbms/lib directory. Therefore it is not in the usual $ORACLE_HOME/bin directory. The linked executable can be moved to the bin directory, or it can be started in the rdbms/lib directory.

```
[oracle@RDBALINUX03 bin]$ bbed
Password: *******

BBED: Release 2.0.0.0.0 - Limited Production on Mon Sep 19 10:00:27 2005

Copyright (c) 1982, 2002, Oracle Corporation.  All rights reserved.

************* !!! For Oracle Internal Use only !!! ***************

BBED>
```

Note that the software will not start unless a password is supplied. This password protects bbed from unauthorized use. The password is hard coded by Oracle and is not dependent on any client settings.

If you know enough about Oracle internals to safely make use of this document and this tool, then you should be able to determine the password yourself using standard reverse-engineering techniques.

Several command-line options can be specified. These allow settings and actions to be scripted. It also allows the bbed mode to be set to browse rather than edit which is advisable for first-time users to ensure that no damage is done to the data files.

The following table shows the command line options available.

Option	Description
BLOCKSIZE	Standard blocksize in bytes of datafiles to edit.

MODE	Mode to run bbed in (browse or edit).
SILENT	Suppresses output to standard out (Y or N).
SPOOL	Spools output to bbed.log file (Y or N).
LISTFILE	List of files to edit.
CMDFILE	Filename of list of commands to execute.
BIFILE	Filename of before-image file (undo) file. Default name is bifile.bbd.
LOGFILE	Filename of user logfile. Default name is log.bbd.
PARFILE	Parameter file with above options listed.

Since several of these options may be required, it is advisable to use the parfile option to store options together in a file. The following is an example of a bbed parfile for a small database with an 8Kb block size.

```
[oracle@pingu bbed]$ cat bbed.par
blocksize=8192
listfile=/home/oracle/bbed/fileunix.log
mode=edit
```

In the above example we have set the block size to 8Kb and set the bbed mode to edit so that we change the data blocks. We have also provided the name of a file that will list all of the files to edit.

The listfile should include the name of the file to edit, the file id of the datafile and the size of the file in bytes. The following is an example from a small sample database:

```
[oracle@pingu bbed]$ cat fileunix.log
1 /home/oracle/OraHome1/oradata/gctdev2/drsys01.dbf 20971520
2 /home/oracle/OraHome1/oradata/gctdev2/example01.dbf 125829120
3 /home/oracle/OraHome1/oradata/gctdev2/indx01.dbf 26214400
4 /home/oracle/OraHome1/oradata/gctdev2/system01.dbf 335544320
5 /home/oracle/OraHome1/oradata/gctdev2/tools01.dbf 10485760
6 /home/oracle/OraHome1/oradata/gctdev2/undotbs01.dbf 246415360
7 /home/oracle/OraHome1/oradata/gctdev2/users01.dbf 26214400
8 /home/oracle/OraHome1/oradata/gctdev2/xdb01.dbf 39976960
```

NB: The listfile can be generated from the target database by running the following command:

```
SQL> select file#||' '||name||' '||bytes from v$datafile
```

Command Overview

Once bbed has started, the user is presented with the bbed prompt. The first command they are likely to use is the help command. To get a listing of help for all available commands, use the command *help all*

```
BBED> help all
SET DBA [ dba | file#, block# ]
SET FILENAME 'filename'
<output removed to aid clarity>
UNDO
HELP [ <bbed command> | ALL ]
VERIFY [ DBA | FILE | FILENAME | BLOCK ]
CORRUPT [ DBA | FILE | FILENAME | BLOCK ]
BBED>
```

The following section describes the commands available. Commands listed with the first letter in braces can be abbreviated using just the first letter. E.g. the *dump* command can be entered as just *d*.

set dba

Sets the current data block using the standard Oracle DBA (Data Block Address) format. This is entered as file_id, block. E.g. Block 632 of file 3 would be accessed as follows:

```
BBED> set dba 3,632
```

If successful, bbed will respond with the RDBA (Relative Data Block Address) of the block accessed:

```
BBED> set dba 3,632
        DBA             0x00c00278 (12583544 3,632)
```

set filename

Sets the current file to the one specified. It must be a valid Oracle data file and it must be enclosed in single quotes. If the file is not in the current path it must also be fully qualified. If successful, bbed will respond showing the file now being accessed.

```
BBED> set filename '/home/oracle/OraHome1/oradata/gctdev2/xdb01.dbf'
        FILENAME        /home/oracle/OraHome1/oradata/gctdev2/xdb01.dbf
```

set file

Sets the current file to the number specified. The number specified must be one of the file ids supplied in the filelist referenced at startup. If successful, bbed will respond showing the file id now being accessed.

```
BBED> set file 2
        FILE#           2
```

set block

Sets the current block. The block is relative to the filename or file already set. The absolute block can be specified, or an offset to the current block can be specified using the plus (+) or (-) symbols. If successful, bbed will respond showing the current block.

```
BBED> set block 16
        BLOCK#          16

BBED> set block +16
        BLOCK#          32
```

set offset

Sets the current offset. The offset is relative to the block already set. The absolute offset can be specified, or an offset to the current offset can be specified using the plus (+) or minus (-) symbols. If successful, bbed will respond showing the current offset.

```
BBED> set offset 20
        OFFSET          20

BBED> set offset -2
        OFFSET          18
```

set blocksize

Sets the blocksize of the current file. The blocksize must match the file selected or an error will be reported. If successful, bbed will respond showing the current blocksize.

```
BBED> set blocksize 8192
        BLOCKSIZE       8192
```

set listfile

Sets the listfile to the specified file. This option can be used if the listfile was not specified on the command line. The listfile must be enclosed in single quotes. If successful, bbed will respond showing the current listfile.

```
BBED> set listfile 'fileunix.log'
        LISTFILE        fileunix.log
```

set width

Sets the current screen width. If not specified bbed will assume an 80-character display. E.g.

```
BBED> set width 132
        WIDTH           132
```

set count

Sets the number of bytes of the data block to display from the dump command. The default is 512. To see an entire 8Kb block therefore you would need to dump the block eight times at offsets 0, 512, 1024, 1536, 2048, 2560, 3092 and 3604. By setting the count higher bbed will dump more of the block each time. By reducing it a smaller dump can be achieved. E.g.

```
BBED> set count 256
        COUNT           256
```

set ibase

Sets the internal number base. The default is decimal. However it can also be set to hexadecimal or octal. This allows the set file, set block and set offset commands to use an alternate base to decimal. If successful, bbed will respond showing the current base:

```
BBED> set ibase hex
        IBASE           Hex

BBED> set block +A
        BLOCK#          11
```

set obase

The purpose of this command is unknown.

set mode

Sets the bbed mode. The options are browse or edit. In browse mode no changes can be made. This is the suggested mode for first-time users, or if you are intending to use the tool only to inspect data blocks.

set spool

This command appears to not be implemented.

show

Shows the current settings of all options. E.g.

```
BBED> show
        FILE#           1
        BLOCK#          1
        OFFSET          0
        DBA             0x00400001 (4194305 1,1)
        FILENAME        /home/oracle/OraHome1/oradata/gctdev2/drsys01.dbf
        BIFILE          bifile.bbd
        LISTFILE        /home/oracle/bbed/fileunix.log
        BLOCKSIZE       8192
        MODE            Browse
        EDIT            Unrecoverable
        IBASE           Dec
        OBASE           Dec
        WIDTH           80
        COUNT           512
        LOGFILE         log.bbd
        SPOOL           No
```

info

Lists the current files being browsed or edited with bbed. E.g.

```
BBED> info
 File#  Name                                                    Size(blks)
 -----  ----                                                    ----------
     1  /home/oracle/OraHome1/oradata/gctdev2/drsys01.dbf             2560
     2  /home/oracle/OraHome1/oradata/gctdev2/example01.dbf           15360
     3  /home/oracle/OraHome1/oradata/gctdev2/indx01.dbf               3200
     4  /home/oracle/OraHome1/oradata/gctdev2/system01.dbf            40960
     5  /home/oracle/OraHome1/oradata/gctdev2/tools01.dbf              1280
     6  /home/oracle/OraHome1/oradata/gctdev2/undotbs01.dbf           30080
     7  /home/oracle/OraHome1/oradata/gctdev2/users01.dbf              3200
     8  /home/oracle/OraHome1/oradata/gctdev2/xdb01.dbf                4880
```

map

The map command shows a map of the current block. It can be combined with the /v option to produce a more verbose output. The map shows the offsets throughout the block where certain information can be found such as the block header, the data block header or the row directory.

If the set commands have not been used to set a current block, or it the user simply wishes to examine another block while keeping the current block their focus, the file name, file id, block or DBA can be specified with the command.

```
BBED> map /v dba 7,12
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 12                                    Dba:0x01c0000c
------------------------------------------------------------
 KTB Data Block (Table/Cluster)

 struct kcbh, 20 bytes                        @0
    ub1 type_kcbh                             @0
    ub1 frmt_kcbh                             @1
< output removed to aid clarity >
 ub4 tailchk                                  @8188
```

By using the information from the map command, the offset can be set to examine or modify the data in the block. A complete breakdown of an Oracle 8+ data block is shown below:

Structure / Element	Description
`struct kcbh, 20 bytes`	*Block Header Structure*
`ub1 type_kcbh`	Block type (see *Header Block Types* below)
`ub1 frmt_kcbh`	Block format 1=Oracle 7, 2=Oracle 8+
`ub1 spare1_kcbh`	Not used
`ub1 spare2_kcbh`	Not used
`ub4 rdba_kcbh`	RDBA - Relative Data Block Address
`ub4 bas_kcbh`	SCN Base
`ub2 wrp_kcbh`	SCN Wrap
`ub1 seq_kcbh`	Sequence number, incremented for every change made to the block at the same SCN
`ub1 flg_kcbh`	Flag: 0x01 New Block 0x02 Delayed Logging Change advanced SCN/seq 0x04 Check value saved – block XOR's to zero 0x08 Temporary block
`ub2 chkval_kcbh`	Optional block checksum (if DB_BLOCK_CHECKSUM=TRUE)
`ub2 spare3_kcbh`	Not used
`struct ktbbh, 72 bytes`	*Transaction Fixed Header Structure*
`ub1 ktbbhtyp`	Block type (1=DATA, 2=INDEX)

union ktbbhsid, 4 bytes	Segment/Object ID
struct ktbbhcsc, 8 bytes	SCN at last block cleanout
b2 ktbbhict	Number of ITL slots
ub1 ktbbhflg	0=on the freelist
ub1 ktbbhfsl	ITL TX freelist slot
ub4 ktbbhfnx	DBA of next block on the freelist
struct ktbbhitl[2], 48 bytes	ITL list index
struct kdbh, 14 bytes	*Data Header Structure*
ub1 kdbhflag	N=pctfree hit(clusters); F=do not put on freelist; K=flushable cluster keys
b1 kdbhntab	Number of tables (>1 in clusters)
b2 kdbhnrow	Number of rows
sb2 kdbhfrre	First free row entry index; -1 = you have to add one
sb2 kdbhfsbo	Freespace begin offset
sb2 kdbhfseo	Freespace end offset
b2 kdbhavsp	Available space in the block
b2 kdbhtosp	Total available space when all TXs commit
struct kdbt[1], 4 bytes	*Table Directory Entry Structure*
b2 kdbtoffs	
b2 kdbtnrow	
sb2 kdbr[1]	*Row Directory*
ub1 freespace[8030]	*Free Space*
ub1 rowdata[38]	*Row Data*
ub4 tailchk	(See *Tailchecks* below)

Different block types are designated by the first byte of the block. The following table shows how to decode the block type:

Header Block Types

ID	Type
01	Undo segment header
02	Undo data block
03	Save undo header
04	Save undo data block
05	Data segment header (temp, index, data and so on)

06	KTB managed data block (with ITL)
07	Temp table data block (no ITL)
08	Sort Key
09	Sort Run
10	Segment free list block
11	Data file header

The last four bytes of all Oracle blocks is the tail check. The following shows how these are structured for an Oracle 9i block:

Tailchecks

The tail of an Oracle 8+ block is a concatenation of the lower order two bytes of the SCN base, the block type and the SCN sequence number. E.g, if the SCN base number is 0x00029728, the block type is 06 and the SCN sequence number is 0x02, the tail check would be 0x97280602:

SCN base	Type	SCN seq
9728	06	02

Although this tail check value is generated from three components, Oracle treats the final value as a single unsigned integer stored as a word (4-bytes). On little-endian architecture machines, which include Intel, the value will be stores as low-order byte first.

Therefore if the tail check is examined in the block using a standard block editor, or the dump command which will be explained in the next section, the byte order may look different. A tail check of 0x97280602 stored on an Intel machine would be written to disk as "02062897".

(d)ump

The dump command dumps the content of the block to the screen. It can be combined with the /v option to produce a more verbose output. The DBA, Filename, File, Block and/or Offset to dump can be specified with the command. If these are not specified the current file, block and offset as established with the set command will be dumped. The size of the dump is limited by the *set count* option and defaults to 512 bytes or alternatively the size of the dump can be specified with the command. The following example shows the first 128 bytes of file 7, block 12, offset 0 being dumped:

```
BBED> dump /v dba 7,12 offset 0 count 128
```

```
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 12      Offsets:    0 to  127  Dba:0x01c0000c
------------------------------------------------------
 06020000 0c00c001 16830300 00000104 l ......À.........
 2e670000 01000000 276c0000 319c0200 l .g......'l..1...
 0000b155 02003200 0900c001 00000000 l ..±U..2...À.....
 00000000 00000000 00000000 00000000 l ................
 00000000 00000000 00000000 00000000 l ................
 00000000 00000000 00000000 00000000 l ................
 00000000 00010100 00001400 721f841f l ............r...
 841f0000 0100ffff 00000000 00000000 l ................

 <16 bytes per line>
```

We can use the Block Header Structure described in the previous section to decode the first 16 bytes of the block as follows:

Type	Format	Unused	RDBA	SCN Base	SCN Wrap	Seq	Flag
06	02	0000	0c00c001	16830300	0000	01	04

(p)rint

The print command allows data structures to be printed in raw or formatted output. The DBA, Filename, File, Block and/or Offset to print can be specified with the command. If these are not specified the current file, block and offset as established with the set command will be printed.

If the print command is issued with just the block and offset to print, bbed will display the data structure at that offset. For example if we printed the contents of File 7, Block 16, Offset 0, bbed would respond that the data structure was the *kcbh* or Data Block Header:

```
BBED> set dba 7,16
        DBA             0x01c00010 (29360144 7,16)

BBED> set offset 0
        OFFSET          0

BBED> p
kcbh.type_kcbh
--------------
ub1 type_kcbh                               @0        0x06
```

It is also possible to use the print command to print individual data structures by specifying the name. The print the data block header for example, we can specify the following:

```
BBED> p kcbh
struct kcbh, 20 bytes                    @0
   ub1 type_kcbh                         @0        0x06
   ub1 frmt_kcbh                         @1        0x02
   ub1 spare1_kcbh                       @2        0x00
   ub1 spare2_kcbh                       @3        0x00
   ub4 rdba_kcbh                         @4        0x01c00010
   ub4 bas_kcbh                          @8        0x000904d6
   ub2 wrp_kcbh                          @12       0x0000
   ub1 seq_kcbh                          @14       0x01
   ub1 flg_kcbh                          @15       0x06 (KCBHFDLC, KCBHFCKV)
   ub2 chkval_kcbh                       @16       0xd444
   ub2 spare3_kcbh                       @18       0x0000
```

If we wanted to determine the number of rows in the block, we could print the data header structure or *kdbh*:

```
BBED> p kdbh
struct kdbh, 14 bytes                    @100
   ub1 kdbhflag                          @100      0x00 (NONE)
   b1 kdbhntab                           @101      1
   b2 kdbhnrow                           @102      9
   sb2 kdbhfrre                          @104     -1
   sb2 kdbhfsbo                          @106      36
   sb2 kdbhfseo                          @108      7839
   b2 kdbhavsp                           @110      7827
   b2 kdbhtosp                           @112      7851
```

We can also specifying certain data structure elements to print such as the row count:

```
BBED> p kdbhnrow
b2 kdbhnrow                              @102      9
```

Note: when printing a data structure, the output is formatted as follows:

Unit Size*	Name	Offset	Value

* Unit size is shown in bytes and indicates if the value is signed (s) or unsigned (u).

In addition to printing information about the specified data structure, the print command can also be used to print information about the location the data structure points to by using the pointer (*) prefix. For example we can display the block row information by printing the *kdbr* data structure:

```
BBED> p kdbr
sb2 kdbr[0]                                 @118      8059
sb2 kdbr[1]                                 @120      8035
sb2 kdbr[2]                                 @122      8009
sb2 kdbr[3]                                 @124      7984
sb2 kdbr[4]                                 @126      7961
sb2 kdbr[5]                                 @128      7937
sb2 kdbr[6]                                 @130      7912
sb2 kdbr[7]                                 @132      7887
sb2 kdbr[8]                                 @134      7839
```

From this we can determine that there are nine rows in this block. Each row pointer requires two bytes and they are stored in the block from offsets 118 through 134. We can then print information about row zero by using *kdbr[0]* as a pointer:

```
BBED> p *kdbr[0]
rowdata[220]
------------
ub1 rowdata[220]                            @8159     0x2c
```

From this we can determine that row zero starts at offset 8159. This can be verified with the dump command described earlier:

```
BBED> d /v dba 7,16 offset 8159 count 16
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 16      Offsets: 8159 to 8174  Dba:0x01c00010
------------------------------------------------------------
 2c000311 44776967 68742045 69736e65 l ,...Dwight Eisne
```

The print command can also print absolute offsets, although it does not offer the count option like the dump command:

```
BBED> p offset 8159
rowdata[220]
------------
ub1 rowdata[220]                            @8159     0x2c
```

The print command defaults to displaying the output in hexadecimal. However it can also be set to display the output in a number of other formats. The following table shows the available formats:

Switch	Display Format
/x	Hex
/d	signed decimal
/u	unsigned decimal
/o	Octal
/c	Character
/n	Oracle Number
/t	Oracle Date
/i	Oracle ROWID

The following shows offset 8163 being printed as hex and a character:

```
BBED> p offset 8163
rowdata[224]
------------
ub1 rowdata[224]                          @8163     0x44

BBED> p /c offset 8163
rowdata[224]
------------
ub1 rowdata[224]                          @8163     D
```

e(x)amine

The examine command is used to display data from the block in raw or formatted output. The DBA, Filename, File, Block and/or Offset to examine can be specified with the command. If these are not specified the current file, block and offset as established with the set command will be examined. If the examine command is issued with just the block and offset to examine, bbed will display the data structure at that offset.

Unlike the print command it cannot interpret data structures, but it can be used to display row information. Combined with knowledge of the data type of the row, it can be used to retrieve complete rows from the block:

The examine command will interpret the data in the block according to the following switches:

Switch	Display Format
/b	b1, ub1 (byte)
/h	b2, ub2 (half-word)
/w	b4, ub4 (word)
/l	b8, ub8 (long) (was b4/ub4 in Oracle7).
/r	Oracle table/index row

The examine command allows switches from the print command to be combined with these specific switches to interpret data. For example if we wanted to interpret data as an Oracle table row with the first column character and the second and third columns numeric, we would execute the command as follows:

```
BBED> x /rcnn
```

The following example shows the print and examine commands being used to step through the first and second rows of a block, with the data interpreted as a row in the format: character, number, number:

```
BBED> p *kdbr[0]
rowdata[220]
------------
ub1 rowdata[220]                          @8159     0x2c

BBED> x /rcnn
rowdata[220]                              @8159
------------
flag@8159: 0x2c (KDRHFL, KDRHFF, KDRHFH)
lock@8160: 0x00
cols@8161:    3

col   0[17] @8162: Dwight Eisnehower
col    1[3] @8180: 1953
col    2[3] @8184: 1961

BBED> p *kdbr[1]
rowdata[196]
------------
ub1 rowdata[196]                          @8135     0x2c

BBED> x /rcnn
rowdata[196]                              @8135
------------
flag@8135: 0x2c (KDRHFL, KDRHFF, KDRHFH)
lock@8136: 0x00
cols@8137:    3
```

```
col   0[12] @8138: John Kennedy
col    1[3] @8151: 1961
col    2[3] @8155: 1963
```

A repeat count can also be specified to repeat the examine command for subsequent rows. The following shows the print command being used to position the offset at the last row and then the next three rows are examined.

```
BBED> p *kdbr[7]
rowdata[48]
-----------
ub1 rowdata[48]                             @7987     0x2c

BBED> x /3rcnn
rowdata[48]                                 @7987
-----------
flag@7987: 0x2c (KDRHFL, KDRHFF, KDRHFH)
lock@7988: 0x00
cols@7989:    3

col   0[13] @7990: George H Bush
col    1[3] @8004: 1989
col    2[3] @8008: 1993

rowdata[73]                                 @8012
-----------
flag@8012: 0x2c (KDRHFL, KDRHFF, KDRHFH)
lock@8013: 0x00
cols@8014:    3

col   0[13] @8015: Ronald Reagan
col    1[3] @8029: 1981
col    2[3] @8033: 1989

rowdata[98]                                 @8037
-----------
flag@8037: 0x2c (KDRHFL, KDRHFF, KDRHFH)
lock@8038: 0x00
cols@8039:    3

col   0[12] @8040: Jimmy Carter
col    1[3] @8053: 1977
col    2[3] @8057: 1981
```

Note that Oracle fills data blocks from the bottom-up, so setting the offset to the first row will prohibit the use of the repeat option. I.e. if the current row is row 3 and a repeat of 2 is specified, rows 3 and 2 will be displayed. If the current row is 7 and a repeat of 4 is specified, rows 7, 6, 5 and 4 will be displayed. Since there is no row lower than row 1 a repeat will result in an error.

(f)ind

The find command is used to locate data within a block. The command allows hex, string or numeric data to be searched for. The pattern can be searched for from the top of the block (offset 0) using the TOP directive, or from the current position using the CURR directive.

Switches are used to determine the data type of the pattern to search for. These are shown below:

Switch	Datatype
/x	Hexadecimal
/d	Decimal
/u	unsigned decimal
/o	Octal
/c	character (native)

Note: Number and Dates are not supported by the find command.

For example, let's say we wanted to search for the string *ar*. We can use the set commands to position bbed over the required block and then search for the string:

```
BBED> set file 7
        FILE#           7

BBED> set block 16
        BLOCK#          16

BBED> set offset 0
        OFFSET          0

BBED> find /c ar TOP
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 16              Offsets: 8048 to 8175           Dba:0x01c00010
------------------------------------------------------------------------
 61727465 7203c214 4e03c214 522c0003 0b476572 616c6420 466f7264 03c2144b
 03c2144e 2c00030d 52696368 61726420 4e69786f 6e03c214 4603c214 4b2c0003
 0e4c696e 646f6e20 4a6f686e 736f6e03 c2144003 c214462c 00030c4a 6f686e20
 4b656e6e 65647903 c2143d03 c214402c 00031144 77696768 74204569 736e6568
```

```
 <32 bytes per line>
```

bbed has located the string *ar* at offset 8048. We can verify this using the dump command:

```
BBED> d /v dba 7,16 offset 8048 count 32
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 16      Offsets: 8048 to 8079  Dba:0x01c00010
------------------------------------------------------------
 61727465 7203c214 4e03c214 522c0003 l arter.Â.N.Â.R,..
 0b476572 616c6420 466f7264 03c2144b l .Gerald Ford.Â.K

 <16 bytes per line>
```

We can see that the first two characters are indeed *"ar"*. If we want to search for the next occurrence of the same pattern, we can just enter the command find with no arguments.

```
BBED> f
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 16               Offsets: 8092 to 8123           Dba:0x01c00010
------------------------------------------------------------------------
 61726420 4e69786f 6e03c214 4603c214 4b2c0003 0e4c696e 646f6e20 4a6f686e

 <32 bytes per line>

BBED> d /v dba 7,16 offset 8092 count 32
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 16      Offsets: 8092 to 8123  Dba:0x01c00010
------------------------------------------------------------
 61726420 4e69786f 6e03c214 4603c214 l ard Nixon.Â.F.Â.
 4b2c0003 0e4c696e 646f6e20 4a6f686e l K,...Lindon John

 <16 bytes per line>
```

copy

The copy command is used to copy blocks from one location to another. As with other commands, the file or filename and offset can be specified, or the DBA can be specified instead. The following example shows block 16 being copied from file 2 to file 1.

Note: The before-image file is purged when this command is used.

```
BBED> copy dba 2,16 to dba 1,16
```

```
Warning: contents of previous BIFILE will be lost. Proceed? (Y/N) y
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (1)
 Block: 16              Offsets:    0 to  127           Dba:0x00400010
------------------------------------------------------------------------
 06020000 1000c001 d99c0200 00000106 d1020000 01000000 276c0000 319c0200
 0000ffbf 02003200 0900c001 08002200 a8000000 746f8000 26003b00 09200000
 d99c0200 00000000 00000000 00000000 00000000 00000000 00000000 00000000
 00000000 00010900 ffff2400 b71e931e 931e0000 09007b1f 631f491f 301f191f

 <32 bytes per line>
```

(m)odify

The modify command is used to change data inside a block. The DBA, Filename, File, Block and/or Offset to modify can be specified with the command. If these are not specified the current file, block and offset as established with the set command will be modified. Alternatively a symbol or symbol pointer can be specified for modification.

The pattern of bytes used to overwrite the original can be specified in hexadecimal, decimal, unsigned decimal, octal or character data using the same switches as the find command.

The following example shows the data at offset 8170 in block 16 of file 7 being modified. The data to modify is character data:

```
BBED> modify /c Eisen dba 7,16 offset 8170
Warning: contents of previous BIFILE will be lost. Proceed? (Y/N) y
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 16              Offsets: 8170 to 8191           Dba:0x01c00010
------------------------------------------------------------------------
 45697365 6e686f77 657203c2 143503c2 143d0106 d99c

 <32 bytes per line>
```

We can then use the dump command to verify the modification:

```
BBED> dump /v dba 7,16 offset 8163 count 64
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 16      Offsets: 8163 to 8191  Dba:0x01c00010
------------------------------------------------------------
 44776967 68742045 6973656e 686f7765 l Dwight Eisenhowe
 7203c214 3503c214 3d0106d9 9c       l r.Â.5.Â.=..Ù.

 <16 bytes per line>
```

assign

The assign command does symbolic assignment, with type and range checking. Either target or source can be omitted for the current offset. For example, the following command assigns structure at current offset to file 4, block 2 ‘s first ITL entry

```
BBED> assign dba 4, 2 ktbbhitl[0]
```

sum

The sum command is used to check and set the block checksum. The DBA, Filename, File, Block and/or Offset to check can be specified with the command. If these are not specified the current file, block and offset as established with the set command will be checked.

The apply directive can be used to update the checksum.

The following example shows the block checksum of file 7, block 16 being checked and then updated:

```
BBED> sum dba 7,16
Check value for File 7, Block 16:
current = 0x02d1, required = 0x09da

BBED> sum dba 7,16 apply
Check value for File 7, Block 16:
current = 0x09da, required = 0x09da
```

push / pop

The push and pop commands are used to push a file, block and offset location onto a memory backed stack and then pop them back. This allows a current location being edited to be temporarily saved while another location is examined or modified.

Note that the stack only stores the location – it does not save the contents.

The following example shows file 7, block 16, offset 8163 being examined. The location is saved with the push command. We then move to file 6, block 1 before returning to DBA 7,16 with the pop command.

```
BBED> push dba 7,16
```

```
        DBA             0x01c00010 (29360144 7,16)
        OFFSET          8163

BBED> set dba 6,1
        DBA             0x01800001 (25165825 6,1)

BBED> pop
        DBA             0x01c00010 (29360144 7,16)
        OFFSET          8163
```

The command *pop all* can be used to remove all push'd entries from the stack. The command *show all* can be used to show all saved locations.

revert

The revert command is used to restore a file, filename, block or DBA to it's original state when bbed was started. E.g.

```
BBED> revert dba 7,12
All changes made to this block will be rolled back. Proceed? (Y/N) y
Reverted file '/home/oracle/OraHome1/oradata/gctdev2/users01.dbf', block 12
```

undo

The undo command rolls back the last modify or assign command. If the undo command is issued again the modification is re-done. The following example shows a modify command being undone:

```
BBED> modify /c Esien dba 7,16 offset 8170
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 16               Offsets: 8170 to 8191           Dba:0x01c00010
------------------------------------------------------------------------
 45736965 6e686f77 657203c2 143503c2 143d0106 d99c

 <32 bytes per line>

BBED> undo
BBED> modify /x 456973656E filename '/home/oracle/OraHome1/oradata/gctdev2/users01.dbf'
block 16. offset 8170.
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 16               Offsets: 8170 to 8191           Dba:0x01c00010
------------------------------------------------------------------------
 45697365 6e686f77 657203c2 143503c2 143d0106 d99c
```

```
 <32 bytes per line>
```

verify

The verify command is used to verify the integrity of the block. It performs a similar function to the *dbverify* utility. The following example show the verify command used on a block we have marked corrupt with the corrupt command:

```
BBED> verify dba 7,12
DBVERIFY - Verification starting
FILE = /home/oracle/OraHome1/oradata/gctdev2/users01.dbf
BLOCK = 12

Block Checking: DBA = 29360140, Block Type = KTB-managed data block
Found block already marked corrupted

DBVERIFY - Verification complete

Total Blocks Examined         : 1
Total Blocks Processed (Data) : 1
Total Blocks Failing   (Data) : 0
Total Blocks Processed (Index): 0
Total Blocks Failing   (Index): 0
Total Blocks Empty            : 0
Total Blocks Marked Corrupt   : 0
Total Blocks Influx           : 0
```

corrupt

The corrupt command is used to mark blocks as media corrupt. For example:

```
BBED> corrupt dba 7,12
Block marked media corrupt.
```

Note: The undo command does not undo a corruption. The revert command however does.

Worked Examples

The following section shows several examples of how to use bbed to modify blocks of an Oracle database.

Note: Although bbed can modify data in the data files of an open Oracle database, it is advisable to shut down the database before making any changes. This avoids the checkpoint process overwriting the changes made with bbed from the Oracle block cache. It also avoids Oracle reading the block before the modifications are complete and declaring the block corrupt.

Important: Using bbed to modify the contents of an Oracle data block renders the data un-supported by Oracle. These examples should be used for educational purposes only. If they are used on real production databases they should only be used as a last resort and once the immediate problem has been resolved, all retrievable data should be exported and a new database created.

Although bbed can be used to open a database that would otherwise be beyond salvaging, the DBA must bear in mind that the internal tables such as OBJ$, UET$ and FET$ may no longer match the contents of the data blocks. The behavior of the database will therefore be unpredictable and ORA-600 errors are likely.

Example #1 - Changing Data

The following example shows how to modify data in an Oracle data file using bbed.

We have determined that there is a typo in our data that needs to be fixed:

```
SQL> select * from presidents;

NAME                                               START_YEAR   END_YEAR
-------------------------------------------------- ---------- ----------
Dwight Eisnehower                                        1953       1961
John Kennedy                                             1961       1963
```

We can determine the block to modify by selecting the ROWID, and then using the ROWID_INFO procedure of the DBMS_ROWID package to determine the file and block:

```
SQL> select rowid from presidents where name = 'Dwight Eisenhower';

ROWID
------------------
AAAGwnAAHAAAAAQAAA

SQL> @lookup_rowid AAAGwnAAHAAAAAQAAA
-----
rowid type:1
```

```
object number:27687
relative fno:7
block number:16
row number:0
-----

PL/SQL procedure successfully completed.
```

The data to change is in file 7, block 16. We can point bbed at this block and then use the find command to locate the offset of the data to change:

```
BBED> set dba 7,16
        DBA             0x01c00010 (29360144 7,16)

BBED> find /c Dwight
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 16               Offsets: 8163 to 8191           Dba:0x01c00010
------------------------------------------------------------------------
 44776967 68742045 6973656e 686f7765 7203c214 3503c214 3d0106d9 9c

 <32 bytes per line>
```

Using the dump command, we can verify the data found, and also locate the exact offset of the data to change. From this we see the string "*Eisnehower*" starts at offset 8170:

```
BBED> dump /v dba 7,16 offset 8163 count 64
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 16      Offsets: 8163 to 8191  Dba:0x01c00010
------------------------------------------------------------
 44776967 68742045 69736e65 686f7765 l Dwight Eisnehowe
 7203c214 3503c214 3d0106d9 9c       l r.Â.5.Â.=..Ù.

 <16 bytes per line>
```

The offending string can now be modified using the modify command. The /c switch tells bbed that the pattern we are using to make the replacement is character data:

```
BBED> modify /c Eisen dba 7,16 offset 8170
Warning: contents of previous BIFILE will be lost. Proceed? (Y/N) y
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 16               Offsets: 8170 to 8191           Dba:0x01c00010
------------------------------------------------------------------------
 45697365 6e686f77 657203c2 143503c2 143d0106 d99c

 <32 bytes per line>
```

We can now verify the changes made by dumping the block again:

```
BBED> dump /v dba 7,16 offset 8163 count 64
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 16      Offsets: 8163 to 8191  Dba:0x01c00010
------------------------------------------------------------
 44776967 68742045 6973656e 686f7765 l Dwight Eisenhowe
 7203c214 3503c214 3d0106d9 9c       l r.Â.5.Â.=..Ù.

 <16 bytes per line>
```

Before Oracle can read the block, the block checksum needs to be updated to reflect the changes made. The sum command is used to check and then reset the block checksum:

```
BBED> sum dba 7,16
Check value for File 7, Block 16:
current = 0x02d1, required = 0x09da

BBED> sum dba 7,16 apply
Check value for File 7, Block 16:
current = 0x09da, required = 0x09da
```

From SQL*Plus we can now see the change:

```
SQL> select * from presidents;

NAME                                               START_YEAR   END_YEAR
-------------------------------------------------- ---------- ----------
Dwight Eisenhower                                        1953       1961
John Kennedy                                             1961       1963
```

Example #2 – Recovering Deleted Rows

The following example demonstrates how to un-delete deleted rows in the block. When rows are deleted in Oracle the data is not actually removed. The row is simply marked as deleted and the free space counters and pointers adjusted accordingly. The status of a row is stored in the Row Header which occupies the first few bytes of each row.

The Row Header consists of the Row Flag, Lock Byte (ITL entry) and Column Count. The first of these – the Row Flag – is a single byte that holds a bitmask that shows the status of the row. The bitmask is decoded as follows:

Cluster Key	Cluster Table Member	Head of row piece	Deleted	First data piece	Last data piece	1st Column continues from previous piece	Last column continues in next piece
128	64	32	16	8	4	2	1

Therefore, columns that fit within a single block, are not chained, migrated or part of a clustered table and are not deleted will have the following attributes:

- Head of Row Piece
- First Data Piece
- Last Data Piece

Therefore the Row Flag is 32 + 8 + 4 = 44 or 0x2c. If this is dumped, using the *alter system dump datafile N block Y* command, this pattern is represented as --H-FL--. E.g.:

```
tab 0, row 7, @0x1ecf
tl: 25 fb: --H-FL-- lb: 0x0  cc: 3
col  0: [13]  47 65 6f 72 67 65 20 48 20 42 75 73 68
col  1: [ 3]  c2 14 5a
col  2: [ 3]  c2 14 5e
```

When a row is deleted the Row Flag is updated and bit 16 is set high. This means that for a typical row the flag value is now 32 + 16 + 8 + 4 = 60 or 0x3c. In the following example we delete the record for Richard Nixon from our table. Then using the bbed find command, we locate the deleted row and dump the contents of the block:

```
BBED> set dba 7,16
        DBA             0x01c00010 (29360144 7,16)

BBED> find /c Nixon TOP
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 16              Offsets: 8096 to 8191           Dba:0x01c00010
------------------------------------------------------------------------
 4e69786f 6e03c214 4603c214 4b2c0003 0e4c696e 646f6e20 4a6f686e 736f6e03
 c2144003 c214462c 00030c4a 6f686e20 4b656e6e 65647903 c2143d03 c214402c
 00031144 77696768 74204569 736e6568 6f776572 03c21435 03c2143d 01062ddd

 <32 bytes per line>
```

We found the string "Nixon" at offset 8096. We need to move back from this offset to find the row header:

```
BBED> dump /v dba 7,16 offset 8080
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 16      Offsets: 8080 to 8191  Dba:0x01c00010
------------------------------------------------------------
 03c2144e 3c02030d 52696368 61726420 l .Â.N<...Richard
 4e69786f 6e03c214 4603c214 4b2c0003 l Nixon.Â.F.Â.K,..
 0e4c696e 646f6e20 4a6f686e 736f6e03 l .Lindon Johnson.
 c2144003 c214462c 00030c4a 6f686e20 l Â.@.Â.F,...John
 4b656e6e 65647903 c2143d03 c214402c l Kennedy.Â.=.Â.@,
 00031144 77696768 74204569 736e6568 l ...Dwight Eisneh
 6f776572 03c21435 03c2143d 01062ddd l ower.Â.5.Â.=..-Ý
```

It would appear that the row starts at offset 8084. We can confirm this by checking the row map, and then printing the value of the row pointer.

```
BBED> p *kdbr[3]
rowdata[121]
------------
ub1 rowdata[121]                            @8084     0x3c
```

The Row Flag value of 0x3c also matches our expectation. We need to set bit 16 low, which will change the value of the flag to 0x2c. We will also need to reset the block checksum before trying to re-read the block with Oracle:

```
BBED> modify /x 2c offset 8084
Warning: contents of previous BIFILE will be lost. Proceed? (Y/N) y
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 16               Offsets: 8084 to 8191           Dba:0x01c00010
------------------------------------------------------------------------
 2c02030d 52696368 61726420 4e69786f 6e03c214 4603c214 4b2c0003 0e4c696e
 646f6e20 4a6f686e 736f6e03 c2144003 c214462c 00030c4a 6f686e20 4b656e6e
 65647903 c2143d03 c214402c 00031144 77696768 74204569 736e6568 6f776572
 03c21435 03c2143d 01062ddd

 <32 bytes per line>

BBED> sum dba 7,16 apply
Check value for File 7, Block 16:
current = 0xb470, required = 0xb470
```

Now that the row flag has been updated, we can restart Oracle and re-read the table:

```
SQL> select * from scott.presidents;
```

```
NAME                                               START_YEAR   END_YEAR
-------------------------------------------------- ---------- ----------
Dwight Eisenhower                                        1953       1961
John Kennedy                                             1961       1963
Lindon Johnson                                           1963       1969
Richard Nixon                                            1969       1974
Gerald Ford                                              1974       1977
Jimmy Carter                                             1977       1981
Ronald Reagan                                            1981       1989
George H Bush                                            1989       1993
```

Example #3 – Uncorrupting a Block

The following example shows how bbed can be used to reset the corrupt-block marker. Although Oracle now supports an official PL/SQL package to repair corrupt blocks, the following is still a useful demonstration of the power of bbed.

The following Oracle error shows corrupt data being encountered:

```
SQL> select * from scott.presidents;
select * from scott.presidents
                    *
ERROR at line 1:
ORA-01578: ORACLE data block corrupted (file # 7, block # 16)
ORA-01110: data file 7: '/home/oracle/OraHome1/oradata/gctdev2/users01.dbf'
```

When Oracle determines that a data block is corrupt, it marks the block as corrupt by setting the block sequence number to 0xff. This can be seen as the *seq_kcbh* attribute of the *kcbh* structure:

```
BBED> set dba 7,16
        DBA             0x01c00010 (29360144 7,16)

BBED> p kcbh
struct kcbh, 20 bytes                       @0
   ub1 type_kcbh                            @0        0x06
   ub1 frmt_kcbh                            @1        0x02
   ub1 spare1_kcbh                          @2        0x00
   ub1 spare2_kcbh                          @3        0x00
   ub4 rdba_kcbh                            @4        0x01c00010
   ub4 bas_kcbh                             @8        0x00000000
   ub2 wrp_kcbh                             @12       0x0000
   ub1 seq_kcbh                             @14       0xff
   ub1 flg_kcbh                             @15       0x04 (KCBHFCKV)
   ub2 chkval_kcbh                          @16       0x6ff4
```

```
   ub2 spare3_kcbh                          @18       0x0000
```

Therefore to reset the corrupt marker, we need to set the block sequence number to a value other than 0xff. The sequence number is stored at offset 14. The following shows the sequence number being reset to 0x01.

```
BBED> modify /x 01 dba 7,16 offset 14
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 16               Offsets:   14 to  525           Dba:0x01c00010
------------------------------------------------------------------------
 0104f46f 00000100 0000276c 00002897 02000000 ffbf0200 32000900 c0010400
```

The sequence number also comprises one component of the tailcheck of the block, which occupies the last 8 bytes. This also needs to be reset for Oracle to recognize the block as valid again. Using bbed we can print the tail check and see that it is 0x000006ff. However when we reset it we must remember that this value is interpreted as a single unsigned integer. On Intel machines therefore, the value is stored low-order byte first as the processor uses a little-endian architecture.

```
BBED> p tailchk
ub4 tailchk                                 @8188     0x000006ff

BBED> modify /x 01060000 dba 7,16 offset 8188
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 16               Offsets: 8188 to 8191           Dba:0x01c00010
------------------------------------------------------------------------
 01060000

 <32 bytes per line>
```

Now that the SCN sequence number and tail check has been reset, the block check sum should be re-calculated and applied.

```
BBED> sum dba 7,16 apply
Check value for File 7, Block 16:
current = 0x6f0a, required = 0x6ff4
```

The database will probably have to be bounced to recognize the modified block, since block caching applies to corrupted blocks as well. Once the database is restarted, the block can be read again:

```
SQL> select * from scott.presidents;

NAME                                 START_YEAR   END_YEAR
------------------------------------ ---------- ----------
Dwight Eisnehower                          1953       1961
John Kennedy                               1961       1963
```

```
Lyndon Johnson                                1963       1969
Richard Nixon                                 1969       1974
Gerald Ford                                   1974       1977
Jimmy Carter                                  1977       1981
Ronald Reagan                                 1981       1989
George H Bush                                 1989       1993
Bill Clinton                                  1993       2001

9 rows selected.
```

In this example we only reset the block corruption marker. We did not address the underlying cause of the corruption. If such a cause existed it would need to be addressed before the block was re-read by Oracle, otherwise the block would be marked as corrupt again.

Example #4 – File Header Reset

The following example shows how bbed can be used to manually reset the file header block of a datafile so that the database can be opened. This can be useful for clients who have experienced a database failure when the database was not in archivelog mode. It can also be useful if there are data files that were left out of the backup and have had to be restored from a much older backup.

The following message tells us we have a file that needs recovery:

```
SQL> startup
ORACLE instance started.

Total System Global Area  235999352 bytes
Fixed Size                   450680 bytes
Variable Size             201326592 bytes
Database Buffers           33554432 bytes
Redo Buffers                 667648 bytes
Database mounted.
ORA-01113: file 7 needs media recovery
ORA-01110: data file 7: '/home/oracle/OraHome1/oradata/gctdev2/users01.dbf'
```

The first thing we need to do is to obtain the current change number of the database from the control files and the change number in the file that needs recovery. The following obtains the change number from the control file:

```
SQL> select CHECKPOINT_CHANGE# from v$datafile;

CHECKPOINT_CHANGE#
```

```
------------------
           234618
```

We can see that the database is on change 234618. Now we can check the file that is marked as needing recovery by checking the V$RECOVER_FILE fixed view:

```
SQL> select change# from v$recover_file;

   CHANGE#
----------
    233467
```

The file header is stored in the first block of the data file. We can use bbed to examine the block and show the block map. The header blocks contain a single data structure – *kcvfh*. Oracle considers four attributes of this data structure when determining if a data file is sync with the other data files of the database:

- kscnbas (at offset 140) – SCN of last change to the datafile.
- kcvcptim (at offset 148) - Time of the last change to the datafile.
- kcvfhcpc (at offset 176) – Checkpoint count.
- kcvfhccc (at offset 184) – Unknown, but is always 1 less than the checkpoint point count.

The first two attributes are stored in the *kcvfhckp* sub-structure. The second two are attributes in their own right. We can use the print command to display them all for the file that requires recovery:

```
BBED> p kcvfhckp

struct kcvfhckp, 36 bytes                   @140
   struct kcvcpscn, 8 bytes                 @140
      ub4 kscnbas                           @140      0x00038ffb
      ub2 kscnwrp                           @144      0x0000
   ub4 kcvcptim                             @148      0x2202381c

BBED> p kcvfhcpc
ub4 kcvfhcpc                                @176      0x00000014

BBED> p kcvfhccc
ub4 kcvfhccc                                @184      0x00000013
```

From this we can see that the SCN of the file is 0x38ffb which is 233467. The change time is 0x2202381c. This data is stored at offsets 140 and 148 which we can see with the dump command:

```
BBED> d /v dba 7,1 offset 140 count 16
```

```
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 1       Offsets:  140 to  155  Dba:0x01c00001
------------------------------------------------------
 fb8f0300 00000000 1c380222 01000000 l û........8."....
```

Note that the numbers are stored in little endian format (the low-order byte of the number is stored in memory at the lowest address) as this example database is running on Linux on an Intel platform.

We now need to verify what change number the other data files are on by checking their data headers. The following are the *kcvfhckp, kcvfhcpc* and *kcvfhccc* structures from the SYSTEM datafile:

```
BBED> p kcvfhckp
struct kcvfhckp, 36 bytes                  @140
   struct kcvcpscn, 8 bytes                @140
      ub4 kscnbas                          @140      0x0003947a
      ub2 kscnwrp                          @144      0x0000
   ub4 kcvcptim                            @148      0x22024500

BBED> p kcvfhcpc
ub4 kcvfhcpc                               @176      0x00000019

BBED> p kcvfhccc
ub4 kcvfhccc                               @184      0x00000018
```

We can see that the SCN on the SYSTEM datafile is 0x3947a or 234618. The change time is 0x22024500. Using the modify command we can change the data file header of the older file:

```
BBED> modify /x 7a940300 dba 7,1 offset 140
Warning: contents of previous BIFILE will be lost. Proceed? (Y/N) y
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 1               Offsets:  140 to  155           Dba:0x01c00001
------------------------------------------------------------------------
 7a940300 00000000 1c380222 01000000

 <32 bytes per line>

BBED> modify /x 00450222 dba 7,1 offset 148
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 1               Offsets:  148 to  163           Dba:0x01c00001
------------------------------------------------------------------------
 00450222 01000000 08000000 ca780000

 <32 bytes per line>

BBED> modify /x 19 dba 7,1 offset 176
```

```
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 1                Offsets:  176 to  191           Dba:0x01c00001
------------------------------------------------------------------------
 19000000 00000000 13000000 00000000

 <32 bytes per line>

BBED> modify /x 18 dba 7,1 offset 184
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (7)
 Block: 1                Offsets:  184 to  199           Dba:0x01c00001
------------------------------------------------------------------------
 18000000 00000000 00000000 00000000

 <32 bytes per line>

BBED> sum dba 7,1 apply
Check value for File 7, Block 1:
current = 0xfc23, required = 0xfc23
```

Now that these four attributes of the data file have been manually updated to match the other data files, the database can opened:

```
SQL> startup
ORACLE instance started.

Total System Global Area  235999352 bytes
Fixed Size                   450680 bytes
Variable Size             201326592 bytes
Database Buffers           33554432 bytes
Redo Buffers                 667648 bytes
Database mounted.
Database opened.
```

Example #5 – Recovering Deleted / Damaged Data

The following example demonstrates the recovery of deleted data using an older copy of the data file. This has potential uses for clients who have deleted or erroneously updated their data, but cannot afford the time required to restore the database to another location and export the table to a dump file.

Connect to the database and delete the data from the table.

```
SQL> delete from scott.presidents;

9 rows deleted.
```

```
SQL> commit;

Commit complete.
```

Prepare a bbed parameter file and file list with the current copy of the data file (with the deleted table) and the one from an older back with the data still intact.

```
[oracle@pingu bbed]$ cat bbed_copy.par
blocksize=8192
listfile=/home/oracle/bbed/fileunix_copy.log
mode=edit

[oracle@pingu bbed]$ cat fileunix_copy.log
1 /home/oracle/OraHome1/oradata/gctdev2/users01.dbf 26214400
2 /gct/oradata/backup2/users01.dbf 26214400
```

From the current database, determine the number and location of the blocks that comprise the table affected:

```
SQL> select owner, segment_name, header_file, header_block, blocks
  2  from dba_segments
  3  where owner = 'SCOTT' and segment_name = 'PRESIDENTS';

OWNER    SEGMENT_NAME   HEADER_FILE HEADER_BLOCK     BLOCKS
-------- -------------- ----------- ------------ ----------
SCOTT    PRESIDENTS               7           11          8
```

So we can see that we need to restore blocks in file 7 from block 11 forward through 8 blocks. However the DBA_SEGMENTS view counts blocks from zero whereas bbed counts them from one. Therefore the bbed block where the table starts is in fact block 12. We can verify this with bbed by printing the ktbbh structure which shows the object id (see the section on the map command) as follows:

```
BBED> set dba 1,11
        DBA             0x0040000b (4194315 1,11)

BBED> p ktbbh
BBED-00400: invalid blocktype (35)
```

Block 11 is empty, however when we check block 12:

```
BBED> set dba 1,12
        DBA             0x0040000c (4194316 1,12)
```

```
BBED> p ktbbh
struct ktbbh, 72 bytes                  @20
   ub1 ktbbhtyp                         @20       0x01 (KDDBTDATA)
   union ktbbhsid, 4 bytes              @24
      ub4 ktbbhsg1                      @24       0x00006c27
      ub4 ktbbhod1                      @24       0x00006c27
   struct ktbbhcsc, 8 bytes             @28

<output removed to aid clarity>

         b2 _ktbitfsc                   @86       0
         ub2 _ktbitwrp                  @86       0x0000
      ub4 ktbitbas                      @88       0x00000000
```

Notice also we are referring to DBA 1,12 rather than DBA 7,12. This is due to the use of the special parameter file where we have included both copies of the file. In this example file 1 is the damaged file and file 2 is the one from the backup with the table intact. We will now copy blocks 12 through 20 from file 2 to file 1:

```
BBED> set offset 0
        OFFSET          0

BBED> copy dba 2,12 to dba 1,12
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (1)
 Block: 12               Offsets:    0 to  511           Dba:0x0040000c
------------------------------------------------------------------------
 06020000 0c00c001 16830300 00000104 2e670000 01000000 276c0000 319c0200
< output removed to aid clarity >
 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

BBED> copy dba 2,13 to dba 1,13
 File: /home/oracle/OraHome1/oradata/gctdev2/users01.dbf (1)
 Block: 13               Offsets:    0 to  511           Dba:0x0040000d
------------------------------------------------------------------------
 06020000 0d00c001 319c0200 00000204 b3af0000 01000000 276c0000 319c0200
< output removed to aid clarity >
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

The process is repeated for all 8 blocks. Once completed the Oracle database is restarted and the table checked:

```
SQL> select * from scott.presidents;
```

```
NAME                                               START_YEAR   END_YEAR
-------------------------------------------------- ---------- ----------
Dwight Eisnehower                                        1952       1960
John Kennedy                                             1960       1963
Lindon Johnson                                           1963       1969
Richard Nixon                                            1969       1974
Gerald Ford                                              1974       1977
Jimmy Carter                                             1977       1981
Ronald Reagan                                            1981       1989
George H Bush                                            1989       1993
Bill Clinton                                             1993       2001
```

Bibliography

The following books, documents and articles were consulted in the preparation of this paper:

Dumps, Crashes and Corruptions – Oracle Class Course Notes
Oracle Corp, December 1999

Unpublished Metalink Note:62015.1 - SUPTOOL: BBED - 7.3.2+ Database Block
Oracle Corp, March 2000

Tuning Oracle at the Block Level; Beginners, Go Away!
Rich Niemiec, TUSC, 2005

A Close Look at Oracle8i Data Block Internals
Dan Hotka, February 2001

The Machinations of Oracle – Five Oracle stories prepared with “The Terlingua Block Viewer for Oracle”.
Terlingua Software Inc, 2004

Metalink Note:1061465.6 - How To Determine The Block Header Size
Oracle Corp, November 1998