IBM DB2® Universal Database™
DB2 Problem Determination Tutorial Series

DB2 Problem Determination Tools

Revision 1.01

Table of Contents

DB2 Problem Determination Tools

About this tutorial

Tutorial objectives

There are many tools and commands that are shipped with the DB2 product that are intended to aid in the process of problem determination and problem source identification (PD/PSI). The primary objective of this tutorial is to introduce some of these tools to you and to provide you with the knowledge required to use them in "real life" scenarios. In some cases, the use of these tools will allow you to resolve problems on your own without further assistance from DB2 Support. In other cases, DB2 Support must still be involved but knowing about the tools and how to use them will assist you in collecting information in an accurate and timely fashion.

Some of the topics that will be covered in this tutorial include the db2trc tracing facility, the db2dart database inspection tool, and the memory debugging facility.

Along with an overview of these tools and commands, examples are used to show you how they work and how they should be used in a given situation.

Audience and assumptions

Prior to taking this tutorial, you should already be familiar with the terminology and functionality of the DB2 product and should be comfortable executing DB2 commands and SQL statements. This includes a familiarity with backup and restore, crash and roll-forward recovery, object creation, and data access. It is also recommended that you take the "Introduction to Problem Determination" tutorial before taking this one.

Pre-tutorial setup

In order to use the examples in this tutorial, you need to create the sample database and several other objects by executing the following commands and SQL statements:

```
db2start
db2sampl
db2 "connect to sample"
db2 "create bufferpool bp2 size 100"
db2 "create tablespace dms1 managed by database using (file 'DMS1' 1000)"
db2 "create tablespace dms2 managed by database using (file 'DMS2' 1000)"
db2 "create table t1 (c1 int, c2 int) in dms1"
db2 "create table t2 (c1 int, c2 int) in dms1 index in dms2"
db2 "create table kelly.t3 (c1 int) in dms1"
db2 "create index ix1 on t1 (c1)"
db2 "create index ix2 on t2 (c1)"
db2 "connect reset"
```


Some of the examples in this tutorial make reference to the "database directory". For the purposes of this tutorial, the database directory is the directory that holds the control files (such as the SQLDBCON and SQLOGCTL.LFH files) for a given database. For example, C:\DB2\NODE0000\SQL00001 in Windows®, and /home/db2inst/db2inst/NODE0000/SQL00001 in UNIX®. Determine what the database directory is for database SAMPLE before moving on.

Also, some of the examples make reference to the "diagnostic directory". This is the directory where the db2diag.log file is located. This can be determined by viewing the DIAGPATH database manager configuration parameter. If this parameter is empty then the defaults are in use: SQLLIB\<INSTANCE> in Windows and sqllib/db2dump in UNIX.

Tutorial conventions used

When a tool or utility is first mentioned, it is shown in **bold** text.

All commands, statements, and their output are shown in a `monospace` font.

In general, examples shown are Windows-based. However, they should work fine in the UNIX environment as well (unless otherwise noted).

Some of the examples in this tutorial show the supported options for some of the utilities. These options may change over time but the basics should always remain the same.

About the author

Kelly Schlamb has worked at the IBM Toronto Laboratory since 1995. His first two years were spent on the customer support team that specialized in the core engine components of the DB2 UDB workstation product. Kelly is currently a developer in the Buffer Pool Services team, a component of the data management group in DB2 UDB development. The main responsibilities of this team include buffer pools and storage management.

You can reach Kelly Schlamb by locating his email address in the IBM Global Directory at http://www.ibm.com/contact/employees/us .

Basic PD/PSI tools

Displaying the service level: db2level command

If it appears that DB2 is not functioning as expected or that there may be a defect in the product, you may need to search the list of known APARs or call DB2 Support for assistance. In either case, it is essential that you know what service level your instance is running at.

This information is available by running the **db2level** command:

```
C:\>db2level
db2level: DB21085I Instance "DB2" uses DB2 code release "SQL07025" with
level identifier  "03060105" and informational tokens "DB2 v7.1.0.65",
"s020616" and "WR21306".
```

Do this now to see what level of DB2 you are using.

The last token in the message, WR21306, contains the PTF number and can be used to identify the FixPak number on the DB2 Technical Support Web site. The token prior to that, s020616, shows the date that the product was built and is used by DB2 service personnel and development when diagnosing problems. In this example, the build date is June 16, 2002.

Generating EDU call stacks: db2nstck and db2_call_stack commands

The term EDU stands for "engine dispatch unit" and refers to a thread (Windows) or process (UNIX) that is doing work on behalf of DB2. The **db2nstck** and **db2_call_stack** commands are used to generate call stacks for the EDUs that are running under an instance. A call stack, also known as a stack traceback, shows the processing path that an EDU is currently in. The function that the EDU is currently executing is at the top of the stack, the function that called that one is below it, and so on.

These commands are mainly used when it appears that the DB2 engine has become "hung". (In most cases, problems that appear to be with DB2 are in reality caused by application problems. For example, a long-running transaction may be holding locks that all other applications are waiting on. All of the applications will appear to be hung but they are really just waiting for one or more locks to be released. Situations like this can usually be identified using DB2's snapshot and event monitor tools. See the "Performance Problem Determination" tutorial for more information on monitoring.)

Operating system commands (such as `iostat` and `ps` in UNIX) can be used to determine if EDUs are actually doing any work. If it appears that none of the EDUs are doing any processing, an engine hang may be a possibility. By generating multiple call stacks for each EDU (with a sufficient amount of time in between -- 1 to 2 minutes for example), you can compare the call stacks to see if the EDUs are in a different processing path from one stack to the next. If all of the EDUs are stuck in the same function that was shown in the previous call stack then there may in

fact be a problem with the DB2 engine. In that case, the call stacks must be provided to IBM so that the hang can be analyzed.

In UNIX, the db2_call_stack command generates call stacks for both single and multi-node instances. The call stacks are placed in files in the diagnostic directory (sqllib/db2dump by default). Each EDU has its own file which is called t*processID.node number*. The files are text-based and can be viewed using any editor. Here is an example of a portion of a call stack on UNIX:

```
*** Start stack traceback ***

0xD02E2B88 semop + 0x94
         [ NO FDPR ] ( = offset 0x1153E8 in library libc.a )
0xD2A245A4 wl_sysv_sem_op__FP13sqlo_waitlisti + 0x50
         [ NO FDPR ] ( = offset 0x36444 in library libdb2sys.a )
0xD2A28264 sqloqwait2 + 0x4FC
         [ NO FDPR ] ( = offset 0x3A104 in library libdb2sys.a )
0xD2DFCBA8 sqle_resync_agent_loop__FP20SQLE_RESYNC_AGENT_CB + 0x614
         [ NO FDPR ] ( = offset 0x153A48 in library libdb2engn.a )
0xD2DFC4B8 sqle_run_resync_agent__FPcUl + 0x54
         [ NO FDPR ] ( = offset 0x153358 in library libdb2engn.a )
0xD2A0F8D0 sqloCreateEDU__FPFPcUl_vPcUlP13SQLO_EDU_INFOPl + 0x48C
         [ NO FDPR ] ( = offset 0x21770 in library libdb2sys.a )
0xD2A0F174 sqloRunGDS__Fv + 0x240
         [ NO FDPR ] ( = offset 0x21014 in library libdb2sys.a )
0xD2A0C164 sqloInitEDUServices__Fv + 0x22C
         [ NO FDPR ] ( = offset 0x1E004 in library libdb2sys.a )
0xD2A33004 sqloRunInstance__FPFv_iPFi_vPPvPiT4 + 0x634
         [ NO FDPR ] ( = offset 0x44EA4 in library libdb2sys.a )
0x100087D8 main + 0x1038

*** End stack traceback ***
```

In Windows, the db2nstck command is used for single-node instances and db2_call_stack is used for multi-node instances. All of the call stacks will be placed into a single file in the diagnostic directory (sqllib/*instance* by default). The db2nstck command tells you the name of the. For example:

```
C:\>db2nstck
The stack dump has been saved in the file C:\SQLLIB\DB2\P2292.000
```

This file is in a binary format. To convert it to readable text, you need the db2xprt tool and the DB2 for Windows .dbg files (the tool and the files might not be available in all installations of DB2). The call stack will look similar to the UNIX one shown above.

List active processes: db2_local_ps and db2stat commands

In UNIX, all of the DB2 processes running under an instance can be displayed using the **db2_local_ps** command:

```
/home/db2inst> db2_local_ps

Node 0
     UID     PID    PPID   C    STIME    TTY  TIME CMD
 db2inst   29868   59654   0 22:15:09      -  0:00 db2sysc
 db2inst  115934   29868   0 22:15:09      -  0:00 db2ipccm
 db2inst  132686   29868   0 22:15:09      -  0:00 db2gds
 db2inst  168192  115934   0 22:20:06      -  0:00 db2agent (SAMPLE)
 db2inst   25836  132686   0 22:15:09      -  0:00 db2srvlst
 db2inst   50294  132686   0 22:20:06      -  0:00 db2loggr (SAMPLE)
 db2inst   52472  132686   0 22:20:06      -  0:00 db2pclnr
 db2inst   81474  132686   0 22:20:06      -  0:00 db2pfchr
 db2inst  109366  132686   0 22:20:06      -  0:00 db2pfchr
 db2inst  110922  132686   0 22:20:06      -  0:00 db2dlock (SAMPLE)
 db2inst  120434  132686   0 22:20:06      -  0:00 db2pfchr
 db2inst  164430  132686   0 22:15:09      -  0:00 db2resyn
```

Note that no processes will be shown if the instance is stopped. Run the db2start command if no processes are listed.

In Windows, all of the DB2 processes running under an instance can be displayed using the **db2stat** command:

```
C:\>db2stat

Environment Strings
 --> DB2INSTANCE=DB2
 --> DB2TEMPDIR=C:\SQLLIB

DB2 Processes
        DB2SYSCS         2292   x8F4
        DB2STAT          2136   x858
```

One thing to note in the Windows case is that because DB2 is thread-based (not process-based), you will only see one process (DB2SYSCS) for all of an instance's EDUs. It is obvious that the same degree of information is not returned in Windows as is returned in UNIX, but it is still useful at times to know the process ID for a running instance. For example, you can use the Task Manager utility to determine the CPU and memory usage for a given process ID.

Try the appropriate command yourself, based on the platform you are using.

Check the validity of a backup image: db2ckbkp command

The **db2ckbkp** utility can be used to test the integrity of a backup image and to determine whether or not the image can be restored. It can also be used to display the metadata stored in the backup header. To see the syntax for this command, execute it without any parameters. Run the command yourself now to see how to use it.

Once you have familiarized yourself with the options available, try it on a backup of the sample database. First, take a full database backup of SAMPLE (noting the backup timestamp) and then run the tool against the resulting backup image.

Example Windows usage:

```
C:\>db2 backup database sample

Backup successful. The timestamp for this backup image is :
20021104225354

C:\>db2ckbkp sample.0\db2\node0000\catn0000\20021104\225354.001

[1] Buffers processed:  ###

Image Verification Complete - successful.
```

Example UNIX usage:

```
/home/db2inst> db2 backup database sample

Backup successful. The timestamp for this backup image is :
20021104225640

/home/db2inst> db2ckbkp
SAMPLE.0.db2inst.NODE0000.CATN0000.20021104225640.001

[1] Buffers processed:  ####

Image Verification Complete - successful.
```

Identifying the log file associated with an LSN: db2flsn command

When working on recovery problems, you may come across log sequence numbers (LSNs) in the db2diag.log file. These LSNs uniquely identify log records within the database log files and might be dumped to the db2diag.log file at crash or roll-forward recovery time.

You usually see LSNs when crash recovery or roll-forward recovery starts (showing the starting point in the log files for the recovery), but you may also see one if a problem is encountered. In that case it will indicate the log record being replayed at the time of the failure.

When investigating a recovery problem, it is usually necessary to collect log files from the failing system. Given an LSN, the **db2flsn** tool can be used to identify the log file that the log record resides in. Note that this tool can only be used on recoverable databases (that is, databases that are enabled for log retain).

In the following example, you will force a crash recovery of a database and then use the db2flsn tool to find the log file for one of the LSNs shown in the db2diag.log file.

As mentioned above, the tool works only for recoverable databases. If you have not already enabled log retain for the database then do so now:

```
C:\>db2 update db cfg for sample using logretain on
DB20000I  The UPDATE DATABASE CONFIGURATION command completed
successfully.
DB21026I  For most configuration parameters, all applications must
disconnect
from this database before the changes become effective.

C:\>db2 backup database sample

Backup successful. The timestamp for this backup image is :
20021104231827
```

Next, connect to the database, create a table, and then kill the instance before committing the create operation:

```
C:\>db2 connect to sample

   Database Connection Information

 Database server        = DB2/NT 7.2.5
 SQL authorization ID   = DB2
 Local database alias   = SAMPLE

C:\>db2 -c- "create table lsntest (c1 int) in userspace1"
DB20000I  The SQL command completed successfully.

C:\>db2stop -kill
SQL1064N  DB2STOP processing was successful.
```

Because the database was not shut down properly, it will have to go through crash recovery when a connection attempt is made against it. Note that because of the way in which the instance was killed, you must terminate the old back-end process first.

```
C:\>db2 terminate
DB20000I  The TERMINATE command completed successfully.

C:\>db2start
SQL1063N  DB2START processing was successful.

C:\>db2 connect to sample

   Database Connection Information

 Database server        = DB2/NT 7.2.5
 SQL authorization ID   = DB2
 Local database alias   = SAMPLE
```

In the db2diag.log file you should now see something like the following crash recovery section:

```
2002-11-04-23.23.37.489000   Instance:DB2   Node:000
PID:596(db2syscs.exe)   TID:1968   Appid:*LOCAL.DB2.021105042337
base_sys_utilities  sqledint   Probe:0   Database:SAMPLE

Crash Recovery is needed.

2002-11-04-23.23.37.549000   Instance:DB2   Node:000
PID:596(db2syscs.exe)   TID:1968   Appid:*LOCAL.DB2.021105042337
recovery_manager  sqlpresr   Probe:1   Database:SAMPLE
```



```
DIA3908W Crash recovery has been initiated.  Lowtran LSN is
"0000003E800C",
Minbuff LSN is "0000003E800C".
```

Remember the Lowtran LSN value (which is "0000003E800C" in the example above) and then execute the db2flsn tool from within the database directory:

```
C:\>cd DB2\NODE0000\SQL00001

C:\DB2\NODE0000\SQL00001>db2flsn 0000003E800C
Given LSN is contained in log file S0000000.LOG
```

As shown in the output, the log record associated with LSN 0000003E800C can be found in log file S0000000.LOG.

Display the contents of a bind file: db2bfd tool

There are times when it is beneficial to examine the contents of a DB2 bind file. For example, a vendor might supply an executable and bind file and you might want to see what kinds of SQL statements the application might execute. Also, consider the case where an errant application is deleting data from a table that is supposed to remain read-only but you aren't sure which application is responsible. By examining the bind files for all of the applications that run against the database, you might be able to find a DELETE statement against the table in question and from there dig further into the application's logic to determine what is going wrong.

The **db2bfd** tool can be used to display these kinds of thing as well as some other information.

Execute the tool without any parameters to see its usage. Try this yourself now.

```
C:\>db2bfd

Usage:   db2bfd [ [-b] [-h] [-s] [-v] ] <filespec>

        Where:   <filespec> is a bind file

        Options: -b = display bind file header
                 -h = display this information
                 -s = display SQL statements
                 -v = display host variable declarations
```

Now try the tool on the db2sampl application's bind file that is shipped with DB2. Use the `-s` option to see what SQL statements are contained within it:

```
Windows:  db2bfd -s C:\SQLLIB\BND\DB2SAMPL.BND
UNIX:     db2bfd -s ~/sqllib/bnd/db2sampl.bnd
```


Next, use the `-v` option to see the corresponding host variables that would have been defined in the source code for db2sampl:

```
Windows:  db2bfd -v C:\SQLLIB\BND\DB2SAMPL.BND
UNIX:     db2bfd -v ~/sqllib/bnd/db2sampl.bnd
```


DB2's database inspection tool: db2dart command

Overview of the db2dart tool

db2dart is DB2's tool for verifying the architectural correctness of databases and the objects within them. It can also be used to display the contents of database control files to extract data from tables that might otherwise be inaccessible.

To display all of the possible options, simply execute the db2dart utility without any parameters. As each help page is displayed, you will be prompted to press Enter to move to the next page.

Some options that require parameters, such as the table space ID, are prompted for if they are not explicitly specified on the command line.

By default, db2dart will create a report file with the name databaseName.RPT. For single-partition databases (DB2 EE), the file is created in the current directory. For multiple-partition databases (DB2 EEE), the file is created under a subdirectory in the diagnostic directory. The subdirectory is called DART####, where #### is the node number.

db2dart accesses the data and metadata in a database by reading them directly from disk. Because of that, db2dart should never be run against a database that still has active connections. If there are connections, db2dart will not know about pages in the buffer pool, control structures in memory, etc. and may report false errors as a result. Similarly, if you run db2dart against a database that requires crash recovery or that has not completed roll-forward recovery, similar inconsistencies might result due to the inconsistent nature of the data on disk.

Inspecting databases, table spaces, and tables

The default behaviour for db2dart is to inspect the entire database. Only the database name must be provided in this case. Try running db2dart yourself against the sample database.

```
C:\>db2dart sample

            <portions of the report have been removed to save space>
                        ____________________________________________

          The requested DB2DART processing has completed successfully!
                     All operation completed without error;
                   no problems were detected in the database.
                        ____________________________________________

                Complete DB2DART report found in: SAMPLE.RPT
```

As the output states, the full db2dart report can be found in the file SAMPLE.RPT. You will also notice that in this case db2dart did not find any problems with the database.

If a database is very large and you are only interested in one table space, you can use the /TS option. When using this option, you must either provide the table space ID on the command line (by specifying the /TSI parameter) or you can let db2dart prompt you for it. For example, to inspect the USERSPACE1 table space (which has a table space ID of 2 in the sample database), either of these commands will work:

```
db2dart sample /ts /tsi 2
db2dart sample /ts          <= When prompted for the table space ID,
enter "2".
```

Similarly, a single table and its associated objects (LOBs, indexes, etc.) can be inspected using the /T option. When using this option, you must provide either the table name or object ID and the ID of the table space in which the table resides.

To determine the object ID and table space ID for a table, you can query the FID and TID columns of the SYSIBM.SYSTABLES catalog table. For example, determine the object ID and table space ID for the EMP_PHOTO table in the sample database by executing the following query:

```
C:\>db2 connect to sample

   Database Connection Information

 Database server        = DB2/NT 7.2.5
 SQL authorization ID   = DB2
 Local database alias   = SAMPLE

C:\>db2 "select creator,name,tid,fid from sysibm.systables where name =
'EMP_PHOTO'"

CREATOR            NAME                  TID    FID
------------------ --------------------- ------ ------
DB2                EMP_PHOTO             2      8

  1 record(s) selected.

C:\>db2 connect reset
DB20000I  The SQL command completed successfully.
```

To inspect this table, execute either of the following db2dart commands:

```
db2dart sample /t /tsi 2 /oi 8
db2dart sample /t                <= When prompted for the table ID and
                                    table space ID, enter "8 2".
```

As mentioned above, the table name can be specified instead of the object ID:

```
db2dart sample /t /tsi 2 /tn EMP_PHOTO
db2dart sample /t                          <= When prompted for the table
                                              name and table space ID,enter
                                              "EMP_PHOTO 2".
```


Dumping formatted table data

If a table space or table becomes corrupt for any reason (for example due to a bad disk or disk controller), attempts to access the table through SQL may not work. (The SQL statement may fail with an error or the database may be marked bad and all connections will be dropped.) In such a case, entries will likely be written to the db2diag.log file, indicating that a bad page was encountered. The db2diag.log entries will contain function names such as **sqlbReadPage**, **sqlbrdpg**, and **sqlbcres**. If you see such entries, you should run db2dart against the database (or table space) to determine the extent of the damage.

If this happens, it may be necessary to extract all of the data possible so that the table space and table can be rebuilt. In such a situation, the /DDEL option of db2dart can be used to extract the table data and place it into a delimited ASCII file. Note that due to the nature of ASCII files, some columns (such as LOB columns) cannot be extracted from the table. db2dart will tell you if this is the case.

When using the /DDEL option, you must provide a table space ID, object ID, starting page number, and number of pages. To extract all of the pages, use 0 for the starting page number and some very large number for the number of pages. (Specifying more pages than actually exist will not cause any problems.)

The ORG table in the sample database resides in table space 2 and has an object ID of 2. To extract all of the data from this table, execute this command:

```
db2dart sample /ddel
```

When prompted, enter either of the following lines of input:

```
2 2 0 1000
ORG 2 0 1000
```

You will then be presented with the column definitions for the table and will be asked to specify an output file name:

```
   Table object data formatting start.
   Please enter
Table ID or name, tablespace ID, first page, num of pages:
(suffic page number with 'p' for pool relative)
2 2 0 1000

   5 of 5 columns in the table will be dumped.
   Column numbers and datatypes of the columns dumped:
         0  SMALLINT
         1  VARCHAR() -VARIABLE LENGTH CHARACTER STRING
         2  SMALLINT
         3  VARCHAR() -VARIABLE LENGTH CHARACTER STRING
         4  VARCHAR() -VARIABLE LENGTH CHARACTER STRING
   Default filename for output data file is TS2T2.DEL,
do you wish to change filename used? y/n
```

You can choose the default or specify a new one. The output file will be created in the current directory be default.

Marking an index object as “bad”

You might encounter a situation where an index on a table has become corrupt. This might result in queries failing or in wrong results being returned. In some cases, the index can be dropped and recreated but it might be easier to mark the index as "bad". What this means is that DB2 will recognize the index as being broken and will rebuild it. The time at which the index gets rebuilt is dependent on the INDEXREC database manager configuration parameter.

The /MI option in db2dart is used to do this. A table space ID and the index object ID specify which index needs to be rebuilt. Note that all indexes for a table are stored in a single index object which means that all indexes will be marked for rebuild.

In most cases, the index object has the same object ID as its base table but this is not always the case. If the index object resides in the same table space as the table then they definitely will have the same ID. However, the IDs can be different if they reside in different table spaces.

To determine the index object ID for a table, inspect that table. For example, inspect tables T1 and T2 in table space DMS1 (which has a table space ID of 3):

```
db2dart sample /t /tsi 3 /tn T1
```

Look for the T1 entry. It should look something like this:

```
   Table inspection start: DB2.T1

      Data inspection phase start. Data obj: 4  In pool: 3
      Data inspection phase end.


      Index inspection phase start. Index obj: 4  In pool: 3
       Scanning pages for index itoken(1) root page:258p.
      Index inspection phase end.

   Table inspection end.
```

As you can see, both the data object (the table) and the index object have the same table space ID (3) and object ID (4). To mark this index object as bad, execute the following db2dart command:

```
db2dart sample /mi /tsi 3 /oi 4
```

If this is successful, you will see the following:

```
Connecting to Buffer Pool Services...
Attempting to mark index (p=3;o=4) as bad.

Modification for page (obj rel 0, pool rel 256) of pool ID (3) obj ID
```



```
(4), written out to disk successfully.
                    ____________________________________________

        The requested DB2DART processing has completed successfully!
                    All operation completed without error;
                  no problems were detected in the database.
                    ____________________________________________
```

When T2 was created, it was created such that its table was in one table space and its indexes were in another. Find the index object ID for this table:

```
db2dart sample /t /tsi 3 /tn T2
```

Look for the T2 entry. It should look something like this:

```
   Table inspection start: DB2.T2

      Data inspection phase start. Data obj: 5  In pool: 3
      Data inspection phase end.


      Index inspection phase start. Index obj: 4  In pool: 4
      Index inspection phase end.

   Table inspection end.
```

In this case, the table space IDs are different (this is expected) and the object IDs are different as well. To mark the index object bad, specify the index table space (4) and index object ID (4):

```
db2dart sample /mi /tsi 4 /oi 4
```

If this is successful, you will see the following:

```
Connecting to Buffer Pool Services...
Attempting to mark index (p=4;o=4) as bad.

Modification for page (obj rel 0, pool rel 128) of pool ID (4) obj ID
(4), written out to disk successfully.
                    ____________________________________________

        The requested DB2DART processing has completed successfully!
                    All operation completed without error;
                  no problems were detected in the database.
                    ____________________________________________
```

If you accidentally use the table space ID and object ID of the base table, it will fail because the index object will not be found:


```
db2dart sample /mi /tsi 3 /oi 5

Connecting to Buffer Pool Services...
Attempting to mark index (p=3;o=5) as bad.

DB2DART Processing completed with error!

              WARNING:
The inspection phase did not complete!
```

Displaying formatted table space file information

Within the database directory you will find two files called SQLSPCS.1 and SQLSPCS.2. These files are copies of each other and contain all information about the table spaces and containers belonging to the database.

The /DTSF option of db2dart can be used to display all of the information in these files. You might want to do this if you are interested in information about a table space that is not available through the `LIST TABLESPACES SHOW DETAIL` command. For example, the quiesce user ID for a table space is not available in the output of that command but it is stored in the SQLSPCS files. Also, the table space map is only available by looking directly at these files. (The table space map is the conversion table that maps logical page numbers in a DMS table space to physical disk locations.)

Try quiescing a table space in the sample database and use the /DTSF option to view the contents of the SQLSPCS.1 files:

```
db2 "connect to sample"
db2 "quiesce tablespaces for table kelly.t3 share"
db2 "connect reset"
db2dart sample /dtsf
```

The report file will contain entries for each table space defined in the database. Each entry will look something like the one below. Note the "quiescer" information in the output:

```
Information for Tablespace ID: 3
------------------------------------

Tablespace name: DMS1
Table space flags (HEX): 0102
Table space type: Database Managed Space (DMS)
Page size: 4096
Extent size: 32
Prefetch size: 16
Version: 5
Tablespace state: 1
Number of quiescers: 1

 1) Userid of quiescer:             DB2
    Quiesce state:                  1
    Tbspace ID of quiesced object:  3
    Table ID of quiesced object:    6
```



```
            EDU ID:                          0
            Agent ID:                        0

          Usable pages in tablespace: 960
          Total pages in tablespace: 1000
          SMP page for first free extent: 32
          SMP page for last allocated tablespace extent.  63
          SMP extent number of the last initialized SMP extent:  0
          High Water Mark: 352
          Number of containers associated with this tablespace: 1
          Container list:
              Total    Useable  Container       Container
          #   Pages    Pages    Type            Name
          === ======== ======== ==============  ============================
            0     1000      960 striped file    C:\DB2\NODE0000\SQL00001\DMS1
          Container checksum for disk space: 827411818
          Number of ranges in the map: 1
          Size of the map: 1
          Map entry size: 52
          Current map:
                        MaxPage   MaxExtent StartStripe   EndStripe  Adj
      Containers
                [ 0]         959          29           0          29    0    1
      (0)
          Map checksum for disk space: 958
```

Before moving on to the next section, reset the quiesce state for table space DMS1:

```
db2 "connect to sample"
db2 "quiesce tablespaces for table kelly.t3 reset"
db2 "connect reset"
```

New DB2 Version 8 INSPECT command

A new online inspection command called INSPECT was introduced in DB2 Version 8. This command allows you to perform similar database, table space, and table checking as is done by db2dart. There are many benefits to using the INSPECT command including the ability to run it while there are active connections against the database. Also, it is built into the engine which means that significant performance gains are achieved through the use of buffer pools and prefetchers.

The INSPECT command is documented in the DB2 Version 8 Command Reference.

DB2's trace facility: db2trc command

Overview of tracing and the db2trc facility

The **db2trc** command controls the trace facility provided with DB2. The trace facility records information about operations and formats this information into a readable form. Please keep in mind that there is added overhead when a trace is running so enabling the trace facility may impact your system's performance.

In general, traces are used by the DB2 support and development teams when debugging customer problems. You might run a trace to gain further information about a problem that you are investigating, but its use is rather limited without the DB2 source code.

In any case, it is important to know how to correctly turn on tracing and how to dump trace files, just in case you need to send one to the DB2 support and development teams.

Trace parameters and turning trace on

To get a general idea of the options available, execute the db2trc command without any parameters:

```
C:\>db2trc
Usage: db2trc (chg|clr|dmp|flw|fmt|inf|off|on) options

        chg|change
                change the trace mask, maxSysErrors or maxRecordSize
        clr|clear
                clear the trace
        dmp|dump
                dump the trace to a binary trace file
        flw|flow
                show control flow of the trace
        fmt|format
                format the trace
        inf|info|information
                get information on the trace
        off
                turn the trace off
        on
                turn the trace on

        For more information type db2trc (chg|clr|dmp|flw|fmt|inf|off|on)
-u
```

To get more information on a specific option, use the `-u` option. For example, for more information on turning trace on, execute the following command:

```
C:\>db2trc on -u
Usage: db2trc on
        [-m <mask>]
                <mask> = <prods>.<events>.<comps>.&ltfncs>
```



```
                <prods> is ignored.
                <events>, <comps>, and lt;fncs> may be comma (,)
separated lists
                and/or hyphan (-) separated ranges, or single entries.
                An askerisk (*) may be used to match anything.
        [-p <pid>[.<tid>]]
                (trace only this proc/thread)
        [-c <cpid>]
                (trace only this companion proc)
        [-rc <rc>]
                (treat rc as a SysError)
        [-e <maxSysErrors>]
                (stop trace after maxSysErrors)
        [-r <maxRecordSize>]
                (truncate records to maxRecordSize bytes)
        [-s | -f <fileName>]
                (send to shared mem, or file)
        [-l [<bufferSize>] | -i [<bufferSize>]]
                (retain the last or the initial records)
        [-crash mask [-passno loop_count] [-sleep sleep_time] ]
                <mask> = <prods>.<events>.<comps>.<fncs>.<tracepoints>
                loop_count: crash/sleep on loop_count iteration only.
                sleep_time: sleep for sleep_time seconds instead of
crashing.
        [-d]
                (check data pointer validity)
```

The most important option that you need to be aware for turning on trace is `-L`. This specifies the size of the memory buffer that will be used to store the information being traced. If the buffer is too small, information might be lost. (By default only the most recent trace information is kept if the buffer becomes full.) If the buffer is too large, it might be difficult to send the file to the DB2 support team.

The size of the buffer is given in bytes. If tracing an operation that is relatively short (such as a database connection), a size of 8000000 (approximately 8MB) might be sufficient:

```
C:\>db2trc on -l 8000000
Trace is turned on
```

However, if you are tracing a larger operation or if a lot of work is going on at the same time, a larger trace buffer might be required. The next panel will discuss how to determine whether the trace buffer chosen is large enough.

On most platforms, tracing can be turned on at any time and works as described above. However, there are things that you need to know before tracing on the Solaris platform. On Solaris, if the trace is turned off after the instance has been started, a very small buffer will be used regardless of the size specified. To effectively run a trace on Solaris, turn the trace on before starting the instance and "clear" it as necessary afterwards (see the next panel for more details on this).

Dumping and formatting a trace

Once the trace facility has been enabled using the on option, all subsequent work done by the instance will be traced.

While the trace is running, you can use the `clr` option to clear out the trace buffer. (All existing information in the trace buffer will be removed.)

```
C:\>db2trc clr
Trace has been cleared
```

Once the operation being traced has finished, use the `dmp` option followed by a trace file name to dump the memory buffer to disk. For example:

```
C:\>db2trc dmp trace.dmp
Trace has been dumped to file
```

The trace facility will continue to run after dumping the trace buffer to disk. To turn tracing off, use the off option:

```
C:\>db2trc off
Trace is turned off
```

The dump file created above is in a binary format that is not readable. You can use the `flw` and `fmt` options to convert this binary file to readable ASCII files. (Descriptions of "formatted" and "flowed" traces will be covered in the next panel.) You must provide the binary dump file along with the name of the ASCII file that you want to create:

```
C:\>db2trc flw trace.dmp trace.flw
Trace wrapped           : NO
Size of trace           : 533244 bytes
Records in trace        : 10667
Records formatted       : 12     (pid: 120; tid: 380; node: 0)
Records formatted       : 1152   (pid: 1528; tid: 1444; node: 0)
...

C:\>db2trc fmt trace.dmp trace.fmt
Trace wrapped           : NO
Size of trace           : 533244 bytes
Records in trace        : 10667
Records formatted       : 10667
```

When you format or flow a binary trace file, the output will tell you whether or not the trace wrapped. If a trace has wrapped, it means that the trace buffer was not large enough to contain all of the information collected during the trace period. The most recent information is maintained by default.

A wrapped trace might be okay depending on the situation. If you are interested in the most recent information then what is in the trace file might be sufficient. However, if you are interested in what happened at the beginning of the trace period or if you are interested in everything that occurred, you might want to redo the operation with a larger trace buffer.

In the following example, a small buffer size is chosen. When the trace is dumped and formatted, we are told that the trace wrapped. The trace is then started again but with a larger buffer. The trace does not wrap the second time.

```
C:\>db2trc on -l 70000
Trace is turned on

C:\>db2 connect to sample

   Database Connection Information

 Database server        = DB2/NT 7.2.5
 SQL authorization ID   = DB2
 Local database alias   = SAMPLE

C:\>db2trc dmp trace.dmp
Trace has been dumped to file

C:\>db2trc off
Trace is turned off

C:\>db2trc fmt trace.dmp trace.fmt
Trace wrapped           : YES
Size of trace           : 64112 bytes
Records in trace        : 1268
Records formatted       : 1268

C:\>db2 connect reset
DB20000I  The SQL command completed successfully.

C:\>db2trc on -l 2000000
Trace is turned on

C:\>db2 connect to sample

   Database Connection Information

 Database server        = DB2/NT 7.2.5
 SQL authorization ID   = DB2
 Local database alias   = SAMPLE

C:\>db2trc dmp trace.dmp
Trace has been dumped to file

C:\>db2trc off
Trace is turned off

C:\>db2trc fmt trace.dmp trace.fmt
Trace wrapped           : NO
Size of trace           : 501368 bytes
Records in trace        : 9909
Records formatted       : 9909
```

Another thing to be aware of is that DB2 will automatically dump the trace buffer to disk when it shuts the instance down due to a severe error. In this case, the file will be created in the diagnostic directory and its name will be db2trdmp.###, where ### is the node number.

Matching up flow and format output

Flowed and formatted traces are different representations of the same thing. One of them may be sufficient to solve a problem but in general they are used together to get a better understanding of what is going on.

The records in a flowed trace are sorted by process or thread, allowing you to follow the code path of a particular EDU. Functions that are executed are listed in an indented fashion that shows the function nesting. Other than return codes from functions, there is little information provided with each function.

The records in a formatted trace are listed chronologically. Some of the trace entries may actually have special information that is important for PD/PSI logged with them.

As an example, let's trace a database connection. (When you try this, the number of trace records will likely not match the number listed below.)

```
C:\>db2trc on -l 8000000
Trace is turned on

C:\>db2 connect to sample

   Database Connection Information

 Database server        = DB2/NT 7.2.5
 SQL authorization ID   = DB2
 Local database alias   = SAMPLE

C:\>db2trc dmp trace.dmp
Trace has been dumped to file

C:\>db2trc off
Trace is turned off

C:\>db2trc flw trace.dmp trace.flw
Trace wrapped           : NO
Size of trace           : 541720 bytes
Records in trace        : 10862
Records formatted       : 1146   (pid: 1668; tid: 304; node: 0)
...

C:\>db2trc fmt trace.dmp trace.fmt
Trace wrapped           : NO
Size of trace           : 541720 bytes
Records in trace        : 10862
Records formatted       : 10862
```

Open the trace.flw file in an editor and search for the function sqlbOpenPTF. You should see something like the section below. From this, you can see that the function sqlbOpenPTF calls function sqloopenp, which in turn calls sqlogmblk.


```
...
5303     | | | | | | | | | | | | | | | |sqlbOpenPTF cei_entry    ...
5304     | | | | | | | | | | | | | | | | |sqloopenp  cei_entry    ...
5305     | | | | | | | | | | | | | | | | |sqloopenp  cei_data     ...
5306     | | | | | | | | | | | | | | | | | |sqlogmblk  cei_entry    ...
5307     | | | | | | | | | | | | | | | | | |sqlogmblk  cei_data     ...
5308     | | | | | | | | | | | | | | | | | |sqlogmblk  cei_data     ...
5309     | | | | | | | | | | | | | | | | | |sqlogmblk  cei_retcode  0
5310     | | | | | | | | | | | | | | | | |sqloopenp  cei_data     ...
5311     | | | | | | | | | | | | | | | | |sqloopenp  cei_errcode  0x1 = 1
5312     | | | | | | | | | | | | | | | | |sqloopenp  cei_entry    ...
5313     | | | | | | | | | | | | | | | | |sqloopenp  cei_data     ...
5314     | | | | | | | | | | | | | | | | | |sqlogmblk  cei_entry    ...
5315     | | | | | | | | | | | | | | | | | |sqlogmblk  cei_data     ...
5316     | | | | | | | | | | | | | | | | | |sqlogmblk  cei_data     ...
5317     | | | | | | | | | | | | | | | | | |sqlogmblk  cei_retcode  0
5318     | | | | | | | | | | | | | | | | |sqloopenp  cei_data     ...
5319     | | | | | | | | | | | | | | | | |sqloopenp  cei_errcode  0x1 = 1
5320     | | | | | | | | | | | | | | | | |sqlbOpenPTF cei_retcode  0
...
```

Open the trace.fmt file in another editor window and look for the trace point numbers listed above (5303 - 5320 in this example). You should see something like this:

```
5303    DB2 cei_entry     buffer_pool_services sqlbOpenPTF (1.20.2.24)
        pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 0
        called_from 00C20A1C

5304    DB2 cei_entry     oper_system_services sqloopenp (1.20.15.97)
        pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 0
        called_from 00C20A44

5305    DB2 cei_data      oper_system_services sqloopenp (1.25.15.97)
        pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 2
        433a 5c44 4232 5c4e 4f44 4530 3030 305c C:\DB2\NODE0000\
        5351 4c30 3030 3031 5c53 514c 5350 4353 SQL00001\SQLSPCS
        2e31 3f00 0000                          .1?...

5306    DB2 cei_entry     oper_system_services sqlogmblk (1.20.15.60)
        pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 0
        called_from 00C20AB8

5307    DB2 cei_data      oper_system_services sqlogmblk (1.25.15.60)
        pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 1
        f8ff 22db 0c00 0000 0000 0000           .."........

5308    DB2 cei_data      oper_system_services sqlogmblk (1.25.15.60)
        pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 250
        a0e8 a201                               ....

5309    DB2 cei_retcode   oper_system_services sqlogmblk (1.21.15.60)
        pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 254
        rc = 0

5310    DB2 cei_data      oper_system_services sqloopenp (1.25.15.97)
        pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 7
        f406 0000                               ....

5311    DB2 cei_errcode   oper_system_services sqloopenp (1.6.15.97)
        pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 254
        rc = 0x1 = 1
```

```
5312    DB2 cei_entry     oper_system_services sqloopenp (1.20.15.97)
        pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 0
        called_from 00C20BB0

5313    DB2 cei_data      oper_system_services sqloopenp (1.25.15.97)
        pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 2
        433a 5c44 4232 5c4e 4f44 4530 3030 305c C:\DB2\NODE0000\
        5351 4c30 3030 3031 5c53 514c 5350 4353 SQL00001\SQLSPCS
        2e32 3f00 0000                          .2?...

5314    DB2 cei_entry     oper_system_services sqlogmblk (1.20.15.60)
        pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 0
        called_from 00C20C24

5315    DB2 cei_data      oper_system_services sqlogmblk (1.25.15.60)
        pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 1
        f8ff 22db 0c00 0000 0000 0000           .."........

5316    DB2 cei_data      oper_system_services sqlogmblk (1.25.15.60)
        pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 250
        c0e8 a201                               ....

5317    DB2 cei_retcode   oper_system_services sqlogmblk (1.21.15.60)
        pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 254
        rc = 0

5318    DB2 cei_data      oper_system_services sqloopenp (1.25.15.97)
        pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 7
        f806 0000                               ....

5319    DB2 cei_errcode   oper_system_services sqloopenp (1.6.15.97)
        pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 254
        rc = 0x1 = 1

5320    DB2 cei_retcode   buffer_pool_services sqlbOpenPTF (1.21.2.24)
        pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 254
        rc = 0
```

From this, you can see that each record has a corresponding entry in both traces. For example:

```
5303    | | | | | | | | | | | | | | | |sqlbOpenPTF cei_entry    ...

  5303   DB2 cei_entry     buffer_pool_services sqlbOpenPTF (1.20.2.24)
         pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 0
         called_from 00C20A1C

5304    | | | | | | | | | | | | | | | |sqloopenp  cei_entry    ...

  5304   DB2 cei_entry     oper_system_services sqloopenp (1.20.15.97)
         pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 0
         called_from 00C20A44

5305    | | | | | | | | | | | | | | | |sqloopenp  cei_data     ...

  5305   DB2 cei_data      oper_system_services sqloopenp (1.25.15.97)
         pid 120; tid 1732; node 0; cpid 1500; msec 87956004; tpoint 2
         433a 5c44 4232 5c4e 4f44 4530 3030 305c C:\DB2\NODE0000\
         5351 4c30 3030 3031 5c53 514c 5350 4353 SQL00001\SQLSPCS
         2e31 3f00 0000                          .1?...
```


You can also see that the sqloopenp function has some information with it in the formatted trace that is not shown in the flowed trace. In this particular case, the formatted trace shows the name of a file that is being opened.

There are thousands of different functions within the DB2 source code that cannot all be explained without access to the source code. However, the sqloopenp function is one that you should be aware of. As you saw in the last example, it indicates that DB2 is trying to open a file.

A problem that will occasionally occur is when somebody accidentally erases or moves a control file for a database and a subsequent connection attempt fails as a result. Depending on the file, this could result in an SQL1036C, SQL5005C, or SQL1042C error.

As an example, let's erase an important database control file for the sample database. The SQLOGCTL.LFH file is a file that keeps track of the database log files and when it is missing an SQL1036C error is returned. This file is located in the database directory. Make sure that there are no connections active against the sample database and then rename the file so that DB2 will not be able to find it:

```
C:\>cd \DB2\NODE0000\SQL00001

C:\DB2\NODE0000\SQL00001>rename SQLOGCTL.LFH SQLOGCTL.BAK

C:\DB2\NODE0000\SQL00001>cd \

C:\>db2 connect to sample
SQL1036C  An I/O error occurred while accessing the database.
SQLSTATE=58030
```

Next, take a trace of the SQL1036C error:

```
C:\>db2trc on -l 8000000
Trace is turned on

C:\>db2 connect to sample
SQL1036C  An I/O error occurred while accessing the database.
SQLSTATE=58030

C:\>db2trc dmp trace.dmp
Trace has been dumped to file

C:\>db2trc off
Trace is turned off

C:\>db2trc flw trace.dmp trace.flw
Trace wrapped           : NO
Size of trace           : 174612 bytes
Records in trace        : 3690
...

C:\>db2trc fmt trace.dmp trace.fmt
Trace wrapped           : NO
Size of trace           : 174612 bytes
Records in trace        : 3690
Records formatted       : 3690
```

Open the trace.flw file and find all occurrences of -1036. You may see more than one but you want to locate the one with the lowest trace entry number on the left hand side (note that this might not be the first occurrence of it in the file). In the output here, the trace entry in question is 2741:

```
2629     | | | | | | | | | | | | |sqlpgint   fnc_entry    ...
2630     | | | | | | | | | | | | | |sqlpgolf   fnc_entry    ...
2631     | | | | | | | | | | | | | | |sqloopenp  cei_entry    ...
2632     | | | | | | | | | | | | | | |sqloopenp  cei_data     ...
2633     | | | | | | | | | | | | | | |sqloopenp  cei_data     ...
2634     | | | | | | | | | | | | | | |sqloopenp  non-fatal_err...
2635     | | | | | | | | | | | | | | |sqloopenp  cei_data     ...
2636     | | | | | | | | | | | | | | |sqloopenp  cei_errcode  0xffffe60a
= -6646
2637     | | | | | | | | | | | | | |sqlpgolf   fnc_errcode  0xffffe60a =
-6646
2638     | | | | | | | | | | | | | |sqltfast2  cei_entry    ...
...
2656     | | | | | | | | | | | | | |sqloopenp  cei_entry    ...
2657     | | | | | | | | | | | | | |sqloopenp  cei_data     ...
2658     | | | | | | | | | | | | | | |sqlogmblk  cei_entry    ...
2659     | | | | | | | | | | | | | | |sqlogmblk  cei_data     ...
2660     | | | | | | | | | | | | | | |sqlogmblk  cei_data     ...
2661     | | | | | | | | | | | | | | |sqlogmblk  cei_retcode  0
2662     | | | | | | | | | | | | | | |sqloflock  cei_entry    ...
2663     | | | | | | | | | | | | | | |sqloflock  cei_data     ...
2664     | | | | | | | | | | | | | | |sqloflock  cei_retcode  0
2665     | | | | | | | | | | | | | | |sqloopenp  cei_data     ...
2666     | | | | | | | | | | | | | | |sqloopenp  cei_errcode  0x1 = 1
...
2738     | | | | | | | | | | | | | |sqltfast2  cei_retcode  0
2739     | | | | | | | | | | | | | |sqlpgint   fnc_errcode  0xffffe60a = -
6646
2740     | | | | | | | | | | | |sqlpinit   cei_errcode  0xffffe60a = -
6646
2741     | | | | | | | | | | |sqledint   fnc_errcode  0xfffffbf4 = -1036
2742     | | | | | | | | | |sqleFirstConnect non-fatal_err...
```

When looking at this entry, you should see sqloopenp functions shortly before it (trace entries 2656 - 2666 in this case). This isn't actually the sqloopenp call that is failing though. If you look at entry 2657 (or whatever the corresponding number is in your case) in the formatted trace file (trace.fmt) you should see this:

```
2657    DB2 cei_data      oper_system_services sqloopenp (1.25.15.97)
        pid 596; tid 1620; node 0; cpid 1308; msec 29982322; tpoint 2
        453a 5c53 514c 4c49 425c 4442 325c 6462 C:\SQLLIB\DB2\db
        3264 6961 672e 6c6f 674b 0200 00        2diag.logK...
```

This is actually DB2 opening the db2diag.log file to write out a "file not found" diagnostic entry to it. The **sqltfast2** function is the function that writes entries to the db2diag.log file so you can ignore the sqloopenp calls that it makes. If you then look for the previous sqloopenp function, you should find it a few trace entries before the sqltfast2 function (entries 2631 - 2636 in the example above):


```
2631     | | | | | | | | | | | | | | |sqloopenp  cei_entry    ...
2632     | | | | | | | | | | | | | | |sqloopenp  cei_data     ...
2633     | | | | | | | | | | | | | | |sqloopenp  cei_data     ...
2634     | | | | | | | | | | | | | | |sqloopenp  non-fatal_err...
2635     | | | | | | | | | | | | | | |sqloopenp  cei_data     ...
2636     | | | | | | | | | | | | | | |sqloopenp  cei_errcode  0xffffe60a
= -6646
```

As you can see, this sqloopenp call failed (0xffffe60a is an internal DB2 error code that represents a "file not found" error). If you look at the corresponding entries in the formatted trace you should see this:

```
2631    DB2 cei_entry     oper_system_services sqloopenp (1.20.15.97)
        pid 596; tid 1620; node 0; cpid 1308; msec 29982322; tpoint 0
        called_from 00BFE83C

2632    DB2 cei_data      oper_system_services sqloopenp (1.25.15.97)
        pid 596; tid 1620; node 0; cpid 1308; msec 29982322; tpoint 2
        433a 5c44 4232 5c4e 4f44 4530 3030 305c C:\DB2\NODE0000\
        5351 4c30 3030 3031 5c53 514c 4f47 4354 SQL00001\SQLOGCT
        4c2e 4c46 485c 0080 00                  L.LFH\...

2633    DB2 cei_data      oper_system_services sqloopenp (1.25.15.97)
        pid 596; tid 1620; node 0; cpid 1308; msec 29982322; tpoint 98
        0200 0000                               ....

2634    DB2 non-fatal_err oper_system_services sqloopenp (1.4.15.97)
        pid 596; tid 1620; node 0; cpid 1308; msec 29982322; tpoint 6
        433a 5c44 4232 5c4e 4f44 4530 3030 305c C:\DB2\NODE0000\
        5351 4c30 3030 3031 5c53 514c 4f47 4354 SQL00001\SQLOGCT
        4c2e 4c46 48                            L.LFH

2635    DB2 cei_data      oper_system_services sqloopenp (1.25.15.97)
        pid 596; tid 1620; node 0; cpid 1308; msec 29982322; tpoint 7
        ffff ffff                               ....

2636    DB2 cei_errcode   oper_system_services sqloopenp (1.6.15.97)
        pid 596; tid 1620; node 0; cpid 1308; msec 29982322; tpoint 254
        rc = 0xffffe60a = -6646
```

From the output above, we see that the file that DB2 is attempting to open is C:\DB2\NODE0000\SQL00001\SQLOGCTL.LFH. As you know, that is the file that we renamed! Problem solved. If you ever encounter this problem in a real life scenario, hopefully the file was just renamed or moved to a different location accidentally. In that case, moving the file back should fix the problem. However, if the file was actually deleted, restoring the database from a backup image is the only supported method of recovery.

Before moving on, rename the file back to its original name and ensure that you can connect to the database again:

```
C:\>cd \DB2\NODE0000\SQL00001

C:\DB2\NODE0000\SQL00001>rename SQLOGCTL.BAK SQLOGCTL.LFH

C:\DB2\NODE0000\SQL00001>cd \

C:\>db2 connect to sample

   Database Connection Information

 Database server        = DB2/NT 7.2.5
 SQL authorization ID   = DB2
 Local database alias   = SAMPLE
```


Mimicking databases: db2look command

Overview of the db2look tool

There are many times when it is advantageous to be able to create a database that is similar in structure to another database. For example, rather than testing out new applications or recovery plans on a production system, it makes more sense to create a test system that is similar in structure and data, and to then do the tests against it instead. This way, the production system will not be affected by the adverse performance impact of the tests or by the accidental destruction of data by an errant application.

Also, when you are investigating a problem (such as invalid results, performance issues, and so on), it may be easier to debug the problem on a test system that is identical to the production system.

You can use the **db2look** tool to extract the required DDL statements needed to reproduce the database objects of one database in another database. The tool can also generate the required SQL statements needed to replicate the statistics from the one database to the other, as well as the statements needed to replicate the database configuration, database manager configuration, and registry variables. This is important because the new database may not contain the exact same set of data as the original database but you may still want the same access plans chosen for the two systems.

The db2look tool is described in detail in the *DB2 Command Reference* but you can view the list of options by executing the tool without any parameters. Try this yourself before moving.

A more detailed usage can be displayed using the `-h` option.

Using db2look to mimic the tables in a database

To extract the DDL for the tables in the database, use the `-e` option. As an example, let's create a copy of the SAMPLE database called SAMPLE2 such that all of the objects in the first database are created in the new database:

```
C:\>db2 create database sample2
DB20000I  The CREATE DATABASE command completed successfully.

C:\>db2look -d sample -e > sample.ddl
% USER is:
% Creating DDL for table(s)
```

Bring up the file sample.ddl in a text editor. Because we want to execute the DDL in this file against the new database, you must change the CONNECT TO SAMPLE statement to CONNECT TO SAMPLE2. While you are at it, take a look at the rest of the contents of the file. You should

see CREATE TABLE, ALTER TABLE, and CREATE INDEX statements for all of the user tables in the sample database:

```
...
------------------------------------------------
-- DDL Statements for table "DB2"."ORG"
------------------------------------------------

 CREATE TABLE "DB2"."ORG"  (
                  "DEPTNUMB" SMALLINT NOT NULL ,
                  "DEPTNAME" VARCHAR(14) ,
                  "MANAGER" SMALLINT ,
                  "DIVISION" VARCHAR(10) ,
                  "LOCATION" VARCHAR(13) )
                 IN "USERSPACE1" ;

------------------------------------------------
-- DDL Statements for table "DB2"."STAFF"
------------------------------------------------

 CREATE TABLE "DB2"."STAFF"  (
                  "ID" SMALLINT NOT NULL ,
                  "NAME" VARCHAR(9) ,
                  "DEPT" SMALLINT ,
                  "JOB" CHAR(5) ,
                  "YEARS" SMALLINT ,
                  "SALARY" DECIMAL(7,2) ,
                  "COMM" DECIMAL(7,2) )
                 IN "USERSPACE1" ;
...
```

Once you have changed the connect statement, execute the statements:

```
C:\>db2 -tvf sample.ddl > sample2.out
```

Take a look at the sample2.out output file -- everything should have been executed successfully. If errors have occurred, the error messages should state what the problem is. Fix those problems and execute the statements again.

As you can see in the output, DDL for all of the user tables are exported. This is the default behaviour but there are other options available to be more specific about the tables included. For example, to only include the STAFF and ORG tables, use the `-t` option:

```
C:\>db2look -d sample -e -t staff org > staff_org.ddl
```

To only include tables with the schema KELLY, use the `-z` option:

```
C:\>db2look -d sample -e -z kelly > kelly.ddl
```


Mimicking statistics for tables

If the intent of the test database is to do performance testing or to debug a performance problem, it is essential that access plans generated for both databases are identical. The optimizer generates access plans based on statistics, configuration parameters, registry variables, and environment variables. If these things are identical between the two systems then it is very likely that the access plans will be the same.

If both databases have the exact same data loaded into them and runstats is performed on both, the statistics should be identical. However, if the databases contain different data or if only a subset of data is being used in the test database then the statistics will likely be very different.

In such a case, you can use db2look to gather the statistics from the production database and place them into the test database. This is done by creating UPDATE statements against the SYSSTAT set of updatable catalog tables as well as RUNSTATS commands against all of the tables.

The option for creating the statistic statements is `-m`. Going back to the SAMPLE/SAMPLE2 example from the previous panel, let's gather the statistics from SAMPLE and add them into SAMPLE2:

```
C:\>db2look -d sample -m > stats.dml
% USER is:
% Running db2look in mimic mode
```

As before, the output file must be edited such that the CONNECT TO SAMPLE statement is changed to CONNECT TO SAMPLE2. Again, take a look at the rest of the file to see what some of the runstats and update statements look like:

```
...
-- Mimic table ORG

RUNSTATS ON TABLE "DB2"."ORG" WITH DISTRIBUTION;

UPDATE SYSSTAT.INDEXES
SET NLEAF=-1,
    NLEVELS=-1,
    FIRSTKEYCARD=-1,
    FIRST2KEYCARD=-1,
    FIRST3KEYCARD=-1,
    FIRST4KEYCARD=-1,
    FULLKEYCARD=-1,
    CLUSTERFACTOR=-1,
    CLUSTERRATIO=-1,
    SEQUENTIAL_PAGES=-1,
    DENSITY=-1
WHERE TABNAME = 'ORG' AND TABSCHEMA = 'DB2';

UPDATE SYSSTAT.COLUMNS
SET COLCARD=-1,
    NUMNULLS=-1
WHERE TABNAME = 'ORG' AND TABSCHEMA = 'DB2';
...
```

As with the `-e` option that extracts the DDL, the `-t` and `-z` options can be used to specify a set of tables.

Generating database layout including node groups, buffer pools, and table spaces

In addition to duplicating the tables and indexes from one database to another, db2look can also be used to generate the definitions for node groups, buffer pools, and table spaces. This is done using the `-L` option. For example:

```
C:\>db2look -d sample -l > structure.ddl
% USER is:
```

Note that the default node groups, buffer pools, and table spaces will not be listed. This is because they already exist in every database by default. If you wish to mimic these, you must alter them yourself manually.

Another thing to note is that if you wish to mimic these types of structures, this must be done before creating the tables and indexes in the test database.

Extracting configuration parameters and environment variables

As mentioned in a previous panel, the optimizer chooses plans based on statistics, configuration parameters, registry variables, and environment variables. As with the statistics, db2look can be used to generate the necessary configuration update and set statements in the test database. This is done using the `-f` option. For example:

```
C:\>db2look -d sample -f > config.txt
% USER is:
```

One thing to note is that only those parameters and variables that affect access plan generation will be included.

Memory debugging

Overview of the memory debug facility

DB2's memory debugging facility is controlled using the DB2MEMDBG registry variable. It enables the tracking of memory allocations including the allocations of memory sets, memory pools, and memory blocks. It can be used to help identify memory corruption or memory leak conditions and to narrow down the point of failure. (Note that it will not completely identify an offending line of code, just help in the process of finding it.)

Keep in mind that DB2 has many different memory heaps and their sizes need to be adjusted based on the workloads being run. As a result, out-of-memory conditions (such as `SQL0956C - Not enough storage in the database heap`) can occur due to poorly tuned configuration parameters and do not necessarily indicate that there is a problem with DB2 itself. Cases where memory debugging might be suggested are when a memory leak is suspected (a static workload is running and memory usage increases over time) or when serious memory errors are encountered (for example, signal #11's in UNIX or db2diag.log entries containing the **sqlofmblk** function).

Memory debugging is not a fully documented process. It is usually done at the request of the DB2 Support team when dealing with potential memory problems and the options used are dependent on the type of problem being investigated. Because one must have an intimate knowledge of DB2's memory architecture and have access to the DB2 source code to fully realize the power of this facility, only the basics will be explained here (just enough so that you will be aware of what it is and be somewhat comfortable with running it if ever asked to).

Enabling memory debug

To enable the memory debugging facility, set the DB2MEMDBG registry variable using the db2set command. Note that the instance must be stopped and started before the variable can take effect.

A common setting for Windows is

```
W:16:0,S:0:655360:4:W:C:A,S:2:655360:4:W:C:A,S:9:5242880:4:W:C:A,
S:10:655360:4:W:C:A,S:11:655360:4:W:C:A
```

This is used for debugging memory corruption in all memory sets. To set this:

```
C:\>db2stop
SQL1064N  DB2STOP processing was successful.

C:\>db2set
DB2MEMDBG=W:16:0,S:0:655360:4:W:C:A,S:2:655360:4:W:C:A,S:9:5242880:4:
W:C:A,S:10:655360:4:W:C:A,S:11:655360:4:W:C:A

C:\>db2start
SQL1063N  DB2START processing was successful.
```

As mentioned before though, the actual value that you will use when debugging a real problem will be given to you by the DB2 support or development analyst, depending on the problem being investigated.

When memory debugging is enabled and the instance is started, a file called memdbg.log will be created in the diagnostic directory. This file contains log entries for all processes that are enabled for memory debugging.

The first portion in the above string (`W:16:0`) causes all memory allocations to be surrounded by a "wall". If this wall ever gets overwritten, that condition will be detected and reported. (Of course, this shouldn't occur if everything is working properly.) Following that are strings like `S:0:655360:4:W:C:A`. Each of these specifies a particular DB2 memory pool (such as the database heap, application heap, private heap, etc). You can actually match each one of these up with the entries in the memdbg.log file.

The equivalent setting for UNIX is

```
W:16,S:0:655360:4:W:O:10:C:A,S:1:655360:4:W:O:10:C:A,S:2:655360:
4:W:O:10:C:A,S:9:5242880:4:W:O:10:C:A,S:10:655360:4:W:O:10:C:A,
S:11:655360:4:W:O:10:C:A,S:12:655360:4:W:O:10:C:A
```

If memory corruption is ever detected or if an out-of-memory condition is encountered, files with the name p*ProcessID*.mem will be created in the diagnostic directory. These files contain per-process memory debugging information, including any information about corruptions that might have occurred.

To turn off memory debugging, unset the registry variable and restart the instance:

```
C:\>db2stop
SQL1064N  DB2STOP processing was successful.

C:\>db2set DB2MEMDBG=

C:\>db2start
SQL1063N  DB2START processing was successful.
```


Summary and Feedback

What you should now know

You should now be familiar with many of the PD/PSI tools that are shipped with DB2, including when they should be used and how they are used.

Feel free to experiment with all of the tools and commands discussed in this tutorial. Unless otherwise noted, they are perfectly safe to run and should not have any adverse affects on your databases. (But of course, all experimentation should be avoided on production databases, just to be safe.)

For more information

Most of the tools and commands described in this tutorial are documented in the DB2 manuals. See the *Command Reference* and the *Troubleshooting Guide* for more information.

Feedback

Please take a moment to give us your thoughts on this tutorial by completing the feedback form on the DB2 Technical Support site at http://www.ibm.com/software/data/db2/udb/winos2unix/support