Benchmark White Paper:
Date: Sept 9, 2003

Integrating PeopleSoft 8.4 GL Batch with IBM DB2 UDB ESE V8.1 (64 Bit) for AIX on IBM eServer pSeries p630 and Network Appliance™ FAS960 Enterprise Server

Authors:

Priti Desai, IBM
Eddie Lee, PeopleSoft
Roger Sanders, Network Appliance, Inc

Table of Contents

1. Summary

This technical paper summarizes benchmark testing that was conducted by IBM, PeopleSoft, and Network Appliance in Pleasanton, CA to measure the Edit (with Combination Editing) and Post processes in PeopleSoft® General Ledger 8.4, using IBM® DB2® Universal Database ESE V8.1 w/FP 2 for AIX on a 4-way IBM® eServer pSeries p630, running IBM® AIX® 5.1 in conjunction with a Network Appliance™ (NetApp®) FAS960 server.

Our results demonstrated excellent functionality between DB2, PeopleSoft, and Network Appliance. We also found that database layout was relatively simple to set up and adjust on the Network Appliance enterprise server, further contributing to ease of use and low cost of ownership.

The focus of the benchmark was to integrate PeopleSoft 8.4 GL on DB2 UDB ESE v8.1 (64 bit) for AIX utilizing a NetApp FAS960 server to accommodate network attached database storage. This document also provides general guidelines on how to setup a DB2 UDB database on a NetApp FAS960 server, as well as information on how to use DB2's Configuration Advisor to achieve optimal performance.

Disclaimer: The material covered in this white paper represents one of the many possible ways to run this benchmark. The results may vary based on your implementation, such as for example Hardware and Software configurations.

a. DB2 UDB ESE, Version 8.1

DB2 is IBM's object-relational database for UNIX*, Linux*, OS/2, and Microsoft® Windows* operating environments. DB2 offers easy installation, integrated functionality, a rich bundle of development tools, full Web enablement, Online Analytical Processing (OLAP) capabilities, and flexibility to scale and change platforms. DB2 provides high performance support for large databases and offers outstanding scalability. For this project we used DB2 UDB ESE V 8.1 for AIX (64-bit) in our testing environment.

IBM has put a lot of research and development effort into providing advanced automation functionality within DB2. The results of this effort are known as SMART ("Self-Managing And Resource Tuning") Technology. The purpose of SMART is to reduce the complexity of managing DB2 for inexperienced users and to reduce the total cost of ownership by enabling many administrative and tuning functions to be carried out automatically by the DBMS. However, unlike some other vendors, IBM has added automation and has not removed the controls required by experienced administrators to fine-tune a database for the highest possible performance.

We used SMART - Performance Configuration Advisor to tune database manager instance and database configuration parameters. We applied all of the Configuration Advisor recommended values except for buffer pool values, which we fine-tuned to achieve highest performance possible.

We took advantage of the DB2 V8.1 64-bit architectures, which allows the creation of larger buffer pools, sort heaps and other memory-intensive resources. Exploiting 64-bit features will also result in performance improvements, as more data can be moved in a single CPU cycle.

b. NetApp FAS960 Enterprise Server

NetApp FAS960 servers provide a simple, yet powerful data management solution for improving performance, ensuring continuous availability, and minimizing infrastructure costs. The flexible, multi-protocol Network Appliance architecture enables simultaneous UNIX file, Windows file, and Web data access to users and application servers across the enterprise. The FAS960 servers offer built-in RAID protection, automatic load balancing, and expanded memory, to deliver maximum scalability and performance for large enterprises. In addition, NetApp servers allow online disk expansion, providing the ability to dynamically add one or more disks to a volume with no application server downtime.

The Network Appliance storage architecture is driven by a robust, tightly coupled, multitasking, real-time micro-kernel called Data ONTAP™. This compact, pre-tuned micro-kernel consists of three primary elements:

- A real-time mechanism that is responsible for processing all incoming requests
- A RAID manager
- The Write Anywhere File Layout (WAFL®) file system

An interface driver within Data ONTAP is responsible for receiving all incoming NFS, CIFS, HTTP, FTP, FCP, and iSCSI requests (depending upon the operating system being used). As each write request is received, it is logged in non-volatile RAM (NVRAM), an acknowledgement is immediately sent back to the requestor, and processing that is needed to satisfy the request is initiated. Once initiated, such processing runs uninterrupted (and continuous), so far as possible. This approach differs from that of traditional file servers, which employ separate processes for handling the network protocol stack, the remote file system semantics, the local file system, and the disk subsystem.

Network Appliance enterprise servers use RAID 4 parity protection for all data stored in the disk subsystem. In the event that any disk drive in the RAID subsystem fails, a "hot spare" disk drive is allocated to that RAID group and data on the failed drive is reconstructed on the hot spare disk drive, using information stored on the parity disk in the RAID group. While reconstruction occurs, requests for data from the failed disk are served by reconstructing the data "on the fly" with no interruption in file service.

The WAFL file system is a UNIX compatible file system that has been optimized specifically for network file access. Network Appliance's WAFL and RAID technologies were designed together to eliminate many of the performance problems that most file systems experience with RAID, and as a result, RAID management is integrated directly into the WAFL file system. By integrating the file system and RAID management, problems that result when RAID management sits on top of the file system (which is how RAID management is usually implemented) are eliminated.

2. Environment

a. Hardware Configuration

The IBM® eServer pSeries model p630 we used was equipped with the following:

- 4 × 1 GHz POWER4® Processors
- 16 Gigabytes of Memory
- 4 Internal disk

- 2 × SSA 160 SerialRAID Adaptor – each connected to Disk Array
- 2 x GB Ethernet-SX PCI Adaptor – each connected to the NetApp FAS960 server
- 2 x Fibre Cable

The Network Appliance FAS960 server we used was configured as follows:

- 256 Megabytes NVRAM
- 6 Gigabytes of Memory (RAM)
- 8 DS-14 disk shelves, each of which consisted of fourteen 72 Gigabyte drives
- 2 x GB Ethernet-SX PCI Adaptors – each connected to the IBM® eServer pSeries model p630

b. Software Configuration

PeopleSoft General Ledger 8.4
PeopleTools 8.42
IBM® DB2® Universal Database Enterprise Server Edition Version 8.1 w/FP 2 for AIX
IBM® AIX® 5.1 Maintenance Level 03 (64-bit)
IBM® WebSphere 4.03
Merant™ Server Express™ (COBOL) 2.0
BEA TUXEDO® 6.5 with Jolt 1.2
NetApp DataONTAP, Version 6.3.1R1

c. Database Layout

We used a NetApp FAS960 server to store the PeopleSoft 8.4 GL database, utilizing a total of 92 disks, each of which had a capacity of 72 Gigabytes. Access to this storage was provided through the mount point **/db2data**. We also used the NetApp FAS960 server to store the transaction log files for our database. In this case, 8 disks were used for storage and access was provided through the mount point **/db2logs**.

Both of the mount points used were created by executing the following commands on the AIX Server:

```
mount 10.1.1.1:/vol/db2data /db2data
mount 10.1.2.1:/vol/db2logs /db2logs
```

Note: We used separate mount points for two volumes on the FAS960 (named **`db2data`** and **`db2logs)`** to provide the ability to replay log transactions in the event that the database should need to be restored as well as to increase the occurrence of parallel I/O between the AIX server and the NetApp FAS960.

The following table illustrates how the mount points we created were used to specify containers for the tablespaces needed by the PeopleSoft 8.4 GL database.

Name	Tablespace Type	Page Size	NFS Mount Point
SYSCATSPACE	SMS Tablespace	4K	/db2data/catalog
TEMPSPACE1	SMS Tablespace	4K	/db2data/temp
USERSPACE1	DMS File Tablespace	4K	/db2data/user1
USERSPACE1IDX	DMS File Tablespace	4K	/db2data/user1
LEDGER	DMS File Tablespace	16K	/db2data/ledger

LEDGERIDX	DMS File Tablespace	16K	/db2data/ledger
TEMP16K	SMS Tablespace	16K	/db2data/tmp16k

Table 1 : Mount points used for benchmark testing

3. Benchmark Scenario

Disclaimer:

The following is an example of the business process breakdown in a typical PeopleSoft 8.4 Financial – General Ledger Application enterprise environment. It may vary based on your H/W configuration and business needs.

a. Data Composition and Volume Scenario

Our benchmark was planned around a data composition model that was designed to measure typical PeopleSoft 8 General Ledger processes, using various data distribution and workload scenarios. These scenarios are representative of typical PeopleSoft customers, categorized as small, medium, large and extra large, according to transaction volumes and historical data required of each respectively sized company. Table 2 outlines this data composition model.

Number of			Small	Medium	Large	Extra Large
Business Units			1	33	33	33
Ledger Rows (2 years)*			705,000	3,525,000	14,100,000	70,920,000
Existing Journal Rows (2 Runs)			100,000	500,000	2,000,000	10,000,000
Journal Transactions per Run			50,000	250,000	1,000,000	5,000,000
Journal Lines per Header	Total for all Accounts	100%	100	250	500	10,000
	Revenue (-)	30%	30	75	150	3000
	Expense (+)	60%	60	150	300	6000
	Asset (+)	6%	6	15	30	600
	Liability (-)	2%	2	5	10	200
	Equity (-)	2%	2	5	10	200
Journal Headers			500	1,000	2,000	5000
Unique ChartFields Combinations			10,000	50,000	200,000	1,000,000
New Unique ChartFields Combinations			5,000	25,000	100,000	500,000
Number of Concurrent Processes			1	7	14	28
Number of Journal Headers per Concurrent Process			500	200	200	25
Ratio of first business Unit versus Number of Concurrent Processes**			N/A	3:7	6:14	12:28
Steps per Group			6	6	6	6

* The value represents the new and existing unique ChartField combinations for 3 ledger types: Local, Budget, and Forecast. The first 23 months represent the number of unique ChartField Combinations. The 24th month contains half of the number of new unique ChartField Combinations (This is done so Journal Post can do 40% Insert and 60% Update).

** The balance of the 32 Business Units will be distributed evenly in the subsequent concurrent processes.

Table 2 : Data Distribution and Volume Scenarios

b. Large EDIT

For this project, we selected the large customer workload from the PeopleSoft Toolkit to process two thousand General Headers, edited in fourteen parallel processes. Table 3 illustrates this workload.

Concurrent Run	Business Unit	Journal Prefix/ Operating Unit	Start #	End #	Journal ID Number	
					From	To
1	B01	P01	101	300	P010000101	P010000300
2	B01	P02	101	300	P020000101	P020000300
3	B01	P03	101	300	P030000101	P030000300
4	B01	P04	101	300	P040000101	P040000300
5	B01	P05	101	300	P050000101	P050000300
6	B01	P06	101	300	P060000101	P060000300
7	B02-B05	P07	101	200	P070000101	P070000200
7B	B06-B09	P07	201	300	P070000201	P070000300
8	B10-B13	P08	101	200	P080000101	P080000200
8B	B14-B17	P08	201	300	P080000201	P080000300
9	B18-B21	P09	101	200	P090000101	P090000200
9B	B22-B25	P09	201	300	P090000201	P090000300
10	B26-B29	P10	101	200	P100000101	P100000200
10B	B30-B33	P10	201	300	P100000201	P100000300

Table 3 : Large EDIT Model

c. Large POST

These two thousand General Headers were posted in ten parallel processes, as shown in Table 4.

Concurrent Run	Business Unit	Journal Prefix/ Operating Unit	Start #	End #	Journal ID Number	
					From	To
1	B01	P01	101	300	P010000101	P010000300
2	B01	P02	101	300	P020000101	P020000300
3	B01	P03	101	300	P030000101	P030000300
4	B01	P04	101	300	P040000101	P040000300
5	B01	P05	101	300	P050000101	P050000300
6	B01	P06	101	300	P060000101	P060000300
7	B02-B09	P07	101	300	P070000101	P070000300
8	B10-B17	P08	101	300	P080000101	P080000300
9	B18-B25	P09	101	300	P090000101	P090000300
10	B26-B33	P10	101	300	P100000101	P100000300

Table 4 : Large POST

d. Assumptions

Note: The PeopleSoft Toolkit makes the following assumptions:

- All PeopleSoft source code will execute unmodified except for routine updates readily available to all customers with the same platform.
- Database table indexes may be added, removed or modified as part of the benchmark.
- One business unit will own 60% of the transactions (i.e., 3,000,000 per run for the largest business unit).

4. Benchmark Server Environment

Our benchmark server environment was based on a logical 4-tier configuration utilizing an IBM eServer pSeries p630 that functioned as a DB Server, an Application Server, and a Web Server. This environment is illustrated in Figure 1.

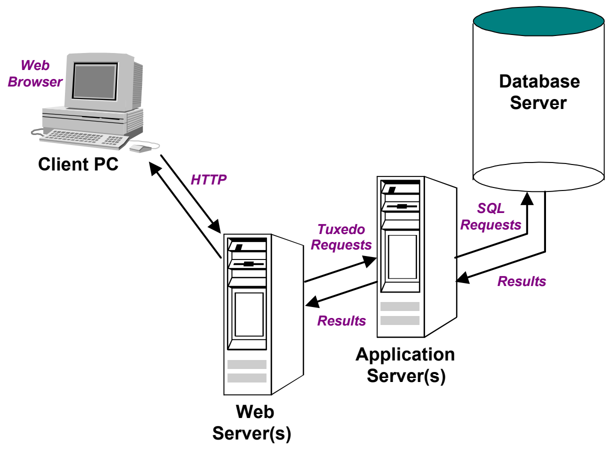

Figure 1 : Logical 4-Tier Environment

a. Database Server

The database server is the backbone of PeopleSoft's Internet Architecture (PIA). It needs sufficient disk space to accommodate the data as well as the database's recovery logs.

For this project, we created the PeopleSoft GL database using DB2 UDB V8.1 ESE, in a 64-bit instance named **db2inst1**.

b. Application Server

The application server is the centerpiece of PeopleSoft's PIA. It connects to the PeopleSoft database and handles almost all SQL-intensive interactions with the database server required during transaction processing. Window-based clients, in three tier configurations, communicate with the application server using Tuxedo messages. In the PIA, the application server interacts with a user workstation via a web server.

For this project, we installed the application server on the same machine as the database server in a 32-bit instance named **db2inst2**.

The application server requires connectivity to the database server. For this project, we used the following commands to establish communications between the application server (running under the

32-bit instance) and a database named GLNETAPP, which was managed by the database server (running under the 64-bit instance).

```
db2 catalog local node sc-ibm10 instance db2inst1
db2 catalog database GLNETAPP as GLNETAPP at node sc-ibm10
db2 connect to GLNETAPP user db2inst1 using db2inst1
```

c. Web Server

A web server is required to run the PeopleSoft Internet Architecture (PIA). The PIA works with any web server that supports Java servlet execution.

For this project, we used IBM WebSphere 4.03 as our web server.

5. Benchmark Storage Environment

Our benchmark storage environment also consisted of a NetApp FAS960 server that was connected to the IBM eServer pSeries p630 using two cross-over fibre cables. This environment is illustrated in Figure 2

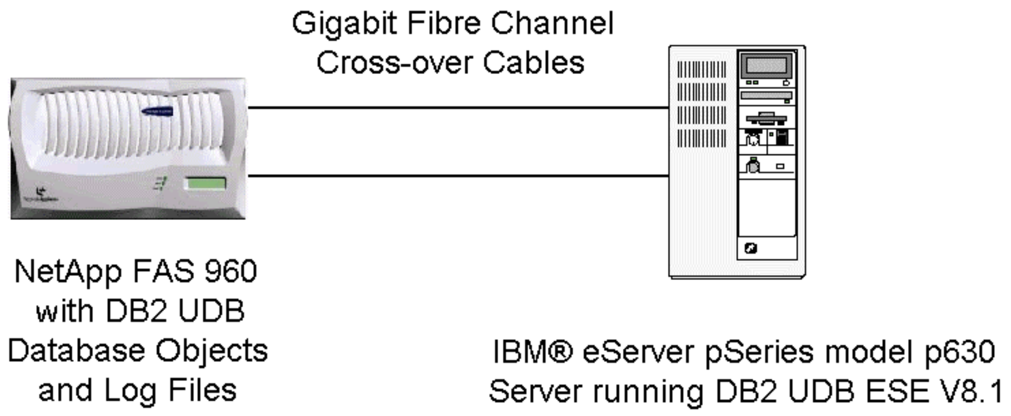

Figure 2 : IBM eServer pSeries p630 – FAS960 Network Environment

a. Physical Disk Layout and Volumes for Data Storage

The Network Appliance Data ONTAP operating system software allows system administrators to create multiple volumes on a NetApp server, each of which may be composed of multiple disks and multiple RAID groups. Each volume in turn contains its own WAFL file system and its own RAID groups. Each volume, in turn, can contain one or more special subdirectories known as *quota trees* or *qtrees*. Qtrees act as virtual sub-volumes that allow a limit to be placed on the size of a directory tree within a NetApp server volume. One may store data directly in a NetApp server volume or in a qtree within a volume. The use of qtrees is not required; however, they may be defined to help organize data. For the purpose of this project we decided not to use qtrees.

Each NetApp server has a special volume, known as the root volume that is used primarily to hold the Data ONTAP operating system. (In fact, the root volume contains a file referred to as */etc/rc*, which contains startup commands that are used by the NetApp server during boot up.) The root volume should be its own volume, composed of at least two disks. It is NetApp's opinion that the extra reliability and flexibility gained by having essentially a mirrored root volume (at least one data disk and one parity disk) outweighs the cost of consuming extra drives. This is especially true in a cluster configuration where a high price is already placed on attaining an extra fraction of a percent of uptime. The root volume usually contains data that doesn't change much over time. A low change rate on the root volume would imply that it does not need to be backed up as often. Should a data volume fail, having a separate, still-functioning root volume will save valuable time in the recovery process. Using a small, two-disk root volume and locating all user data on other volumes makes the job of physically moving foreign volumes between two NetApp servers easier. Furthermore, it is strongly recommended that all the database object files be stored on a separate volume(s) on the NetApp server that is not the root volume. Configuring your filer in this manner provides the capability to restore the database and/or logs using SnapRestore® without affecting the operating system, which is located on its own volume.

For the purpose of backup and recovery using the NetApp server's Snapshot™ technology, a database's transaction log files should be stored on a separate server volume(s) from the volume on which the database's data resides. If the database's log files and data files reside on the same server volume, the recovery of the volume would overwrite existing log files, effectively destroying all log files created after the Snapshot was taken. Thus, the log data needed to perform a roll forward recovery operation would no longer be available.

The NetApp server volumes used in our benchmark testing are illustrated in Table 5.

Volume	Description	RAID Group Size *	Number of Disks
/vol/vol0	Root volume	8	4
/vol/db2data	DB2 database	8	92
/vol/db2log	DB2 logs	8	8
* RAID Group Size value will vary depending upon the optimization routines used in a particular version of Data ONTAP, as well as the size of the NetApp server disk drive being used. The default value provided is usually the optimum size to use.			

Table 5 : NetApp server volumes used for benchmark testing

b. Migrating a DB2 Database to a NetApp Server Using a Redirected Restore

For this project, we migrated our PeopleSoft 8.4 GL database from local SSA disk to the NetApp FAS960 server. The easiest way to physically move a DB2 UDB database from one storage location to another is by backing up the database with DB2's **BACKUP DATABASE** command and then performing what is known as a "redirected restore" operation to copy the database to its new location. Redirected restore operations are performed by executing a combination of DB2's **RESTORE DATABASE** and **SET TABLESPACE CONTAINERS** commands once a backup image of the database has been obtained. The steps needed to perform this type of DB2 database migration are:

1. Establish a connection to the database to be migrated by issuing the following command:

   ```
   $ db2 "CONNECT TO [DB_NAME]"
   ```

   where DB_NAME is the alias assigned to the database when it was created (if no alias was specified, the alias will be the same as the name assigned to the database).

2. List the tablespaces that have been created for the database to be migrated by executing the following command:

   ```
   $ db2 "LIST TABLESPACES"
   ```

3. Record the information obtained about the database's tablespaces and set it aside for later use.

4. Disconnect from the database to be migrated by executing the following command:

   ```
   $ db2 "DISCONNECT [DB_NAME]
   ```

   where DB_NAME is the alias assigned to the database that was created.

5. Stop and restart the DB2 Database Manager instance by executing the following commands:

   ```
   $ db2stop
   $ db2start
   ```

   Note: If the DB2 Database Manager cannot be stopped because one or more database connections exist and if you are confident the database is in a consistent state, you can force all database connections to be terminated before the DB2 Database Manager instance is stopped and restarted by executing the following command:

   ```
   $ db2 "FORCE APPLICATIONS ALL"
   ```

6. Create a full backup image of the database to be moved by issuing the following command:

   ```
   $ db2 "BACKUP DATABASE [DB_NAME] TO [BKUP_LOCATION]"
   ```

   where DB_NAME is the alias assigned to the database that is to be backed up (if no alias has been assigned, the alias will be the same as the name assigned to the database) and BKUP_LOCATION is the location where the database backup image is to reside.

7. Create a script file that contains the commands needed to perform a redirected restore. This script file should contain the following commands:

   ```
   $ db2 "RESTORE DATABASE [OLD_DB_NAME] FROM [BKUP_LOCATION]
   TAKEN AT [BKUP_TMSTAMP] TO [NEW_LOCATION] INTO [NEW_DB_NAME]
   REDIRECT"
   ```

   where: OLD_DB_NAME is the alias assigned to the database that was backed up, BKUP_LOCATION is the location where the database backup image resides, BKUP_TMSTAMP is the timestamp that was assigned to the database backup image (this value can be obtained by examining the file name assigned to the backup image), NEW_LOCATION is the location where the overhead files associated with the database are to reside, and NEW_DB_NAME is the name that is to be assigned to the new database that will be created.

```
$ db2 "SET TABLESPACE CONTAINERS FOR [TSPACE_ID] USING (PATH
‘[DIRECTORY]')"
```

or

```
$ db2 "SET TABLESPACE CONTAINERS FOR [TSPACE_ID] USING (FILE
'[FILE_NAME]' [SIZE])"
```

or

```
$ db2 "SET TABLESPACE CONTAINERS FOR [TSPACE_ID] USING (DEVICE
'[DEVICE_NAME]' [SIZE])"
```

where: TSPACE_ID is the unique identifier (0, 1, 2, etc.) that has been assigned to the tablespace that new container information is to be assigned for, DIRECTORY is the new directory SMS tablespace data is to be stored in, FILE_NAME is the name of the file that DMS tablespace data is to be stored in, DEVICE_NAME is the name of the raw device that DMS tablespace data is to be stored in, and SIZE is the size, in 4, 8, 16, or 32 k pages, of the file or device specified.

Note: Using the information collected in Step 3, you will need to create one of these commands for each tablespace found in the database.

```
$ db2 "RESTORE DATABASE [OLD_DB_NAME] CONTINUE"
```

where OLD_DB_NAME is the alias assigned to the database that was backed up.

For example, the following commands would be used to create a database named new_db on a NetApp server by performing a redirected restore of the database old_db:

```
$ db2 "RESTORE DATABASE old_db FROM /export/home/backup TAKEN AT
20020716102533 TO /mnt/db2nes/db2data/database INTO new_db REDIRECT"

$ db2 "SET TABLESPACE CONTAINERS FOR 0 USING (PATH
'/mnt/db2nes/db2data/system’)"

$ db2 "SET TABLESPACE CONTAINERS FOR 1 USING (FILE
’/mnt/db2nes/db2data/file.dms' 2500)"

$ db2 "SET TABLESPACE CONTAINERS FOR 2 USING (DEVICE '/dev/rusrtbl'
2500)"

$ db2 "RESTORE DATABASE old_db CONTINUE"
```

8. Execute the script file just created to perform a redirected restore operation.

9. Verify that the database was successfully migrated by performing a few simple tests.

For this project, we used the following script to move the PeopleSoft 8.4 GL database from local SSA disk to the NetApp FAS960 server:

```
create database GLNETAPP on /db2data/db_dir collate using identity;

restore database GL842LG2 from /glbkup3/gl842lg2_led2 into GLNETAPP with 4 buffers buffer 512
replace existing redirect parallelism 4;

set tablespace containers for 0 ignore rollforward container operations using (path '/db2data/catalog');

set tablespace containers for 1 ignore rollforward container operations using (path '/db2data/temp');

set tablespace containers for 2 ignore rollforward container operations using (FILE
'/db2data/user1/userdata1.f1' 4718592);

set tablespace containers for 3 ignore rollforward container operations using (FILE
'/db2data/user1/useridx1.f1' 4718592);

set tablespace containers for 4 ignore rollforward container operations using (FILE
'/db2data/ledger/ledgeridx.f1' 655360);

set tablespace containers for 5 ignore rollforward container operations using (FILE
'/db2data/ledger/ledgerdata.f1' 655360);

set tablespace containers for 6 ignore rollforward container operations using (path '/db2data/tmp16k');

restore db GL842LG2 continue;
```

The location where database log files are to be written to can be changed by assigning a value to the database configuration file parameter *newlogpath*. After the PeopleSoft 8.4 GL database was successfully transferred to the NetApp FAS960 server, we made the transferred database store its log files on the NetApp server volume /db2logs by executing the following script:

```
# Change log file location
db2 connect to GLNETAPP
db2 update db cfg for GLNETAPP using NEWLOGPATH /db2logs
db2 force applications all
db2stop
db2start
db2 connect to GLNETAPP

# Verify path to log files are has changed (Check path to log files)
db2 get db cfg for GLNETAPP
```

6. Database Tuning

a. Configuration Advisor – SMART Technology

It was mentioned earlier that DB2 UDB provides two sets of configuration parameters (one for the DB2 Database Manager instance and one for the database itself) that can be used to tune a specific database for optimum performance. What was not mentioned is that a database that uses the default database configuration parameter values provided will not perform as well as a database whose configuration parameters have been even moderately tuned. So if your experience with DB2 UDB is limited, how do you begin to tune your database by modifying the values of these configuration parameters? DB2 UDB, Version 8.1 comes packaged with a tool to help you get started. This tool is called the Configuration Advisor. The Configuration Advisor is designed to capture specific information about your database environment and recommend changes to configuration parameters based upon the information provided. Because configuration changes produced by the Configuration Advisor will almost always result in performance improvement, this tool should be used before any other configuration tuning attempts are made.

Since Configuration Advisor provided with DB2 UDB ESE V8.1 FixPack 2 is not designed to optimize the DB2 Database Manager (DBM) and database (DB) configuration for PeopleSoft Applications, we developed a sample application that calls the DB2 Configuration Advisor API db2AutoConfig() for this project. This application passes the db2AutoConfig() API the set of input parameter values shown in table 6 and updates both DBM and DB settings with any configuration changes recommended. The code for the sample application we used for this project can be seen in Appendix. B.

Input Parameter	Value Used
`memory percentage`	25
`workload`	2 = mixed
`average statement`	1000
`transactions per minute`	1000
`admin priority`	2 = performance
`isPopulated`	1 = yes
`local apps`	2
`remote apps`	14
`isolationLevel`	3 = CS
`bp resizeable`	1 = yes
`client product`	7 = PeopleSoft Batch
`Apply`	1 = yes DB and DBM

Table 6 : db2AutoConfig() API struct values used for this project

In our benchmark, we left all of the sample application configuration changes alone except the following:

```
Bufferpool                    | Old Values | Recommended
-----------------------------------------------------
IBMDEFAULTBP                  |     300000 |      605004
```

Rather than use the recommended value for the IBMDEFAULTBP buffer pool, we chose to allocate 300000 pages to achieve optimal performance.
The actual configuration used in our benchmark environment can be seen in Appendix A.
The sample application that calls the db2AutoConfig API can be seen in Appendix B.

Note: It is relatively simple to modify the sample application to take the input from the command line.

7. The Benchmark Results

The table below indicates the benchmark results for this project. The anticipated results calculation is based on tpm-C and results from a similar benchmark that was run on an IBM eServer pSeries p660 – 8 way server. As Table 7 illustrates, we achieved or surpassed our anticipated results.

Measurement	IBM eServer pSeries p660 – 8 way Benchmark Results	IBM eServer pSeries p630 – 4 way Anticipated Results (40 % of p660)	IBM eServer pSeries p630 - 4 way Actual Results
tpm-C	105 K	43 K	43 K
Edit Journal	2.5 = 150 sec	375 sec	5.77 = 346 sec
Post Journal	1.08 = 65 sec	162 sec	3.00 = 180 sec
Total	3.58 min	9 min	8.77 min
Journal Lines/Hr	16,744,186	6,666,666	6,841,505

Table 7 : Benchmark Anticipated and Actual Results

The testing was conducted in a controlled environment with no other applications running on the server. Performance may vary on other hardware and software platforms and with other data composition models.

8. Backup and Restore with a NetApp Server

The performance and reliability of backup-and-restore operations are critical to effective database administration and NetApp enterprise servers provide unique functionality that offer significant reliability and performance benefits in this area. The key feature of Network Appliance's approach to database backup and recovery is the use of its Snapshot technology. Essentially, a NetApp Snapshot is a read-only image of an entire NetApp server volume's file system, which in our case contained the benchmark database files. You can think of a Snapshot as a "freeze frame" version of a volume for a given point in time. When a Snapshot is created, disk blocks containing data are not copied, but rather, every block utilized in the volume's file system is marked as belonging to the Snapshot. And because these blocks are not overwritten as long as they belong to a Snapshot, the volume's file system can be returned to the state it was in at the time the Snapshot was taken. (For more information on Network Appliance Snapshot technology, refer to the Technical Report titled *File System Design for an NFS File Server Appliance* by Dave Hitz, James Lau, and Michael Malcolm (http://www.netapp.com/tech_library/3002.html#i34).

By using NetApp Snapshot copies, we were able to completely drop and restore our benchmark database (approximately 22 Gigabytes in size) in less than two minutes. (Since there is no copying of data involved when NetApp Snapshot copies are used, an incredible amount of time is saved in returning a file system to its previous state.) The steps we used to backup the database were:

1. Close all connections to the database and stop the DB2 Database Manager instance by issuing the following command:

   $ db2stop force

2. Create a NetApp Snapshot of the volume that contains the data files for the GLNETAPP database by executing the following command from the NetApp server:

   snap create db2data test_bkup

The steps we used to drop and restore the database were:

1. If the DB2 Database Manager instance has not already been started, start it by executing the following command:

   ```
   $ db2start
   ```

2. Drop the benchmark database by executing the following command:

   ```
   $ db2 “DROP DATABASE GLNETAPP”
   ```

3. Verify that the benchmark database no longer exists by issuing the following command:

   ```
   $ db2 "LIST DATABASE DIRECTORY"
   ```

   When this command executes, an entry for the database GLNETAPP should no longer be present.

4. Stop the DB2 Database Manager instance by issuing the following command:

   ```
   $ db2stop
   ```

5. Return the file system of the NetApp server volume that contains the data files for the GLNETAPP database to the state it was in at the time the Snapshot was taken by executing the following command from the NetApp server:

   ```
   snap restore –f –s test_bkup db2data
   ```

   Note: The **snap restore** command is part of SnapRestore, which is a Network Appliance product. SnapRestore requires a separate license code that must be purchased and used to activate the SnapRestore product on the NetApp server before the snap restore command can be used.

6. Restart the DB2 Database Manager instance by executing the following command:

   ```
   $ db2start
   ```

7. Catalog the database that was recreated using the NetApp Snapshot by executing the following command:

   ```
   $ db2 “CATALOG DATABASE GLNETAPP ON /db2data/db_dir”
   ```

8. Verify that the benchmark database has been restored by issuing the following set of commands:

   ```
   $ db2 "LIST DATABASE DIRECTORY”
   $ db2 "CONNECT TO GLNETAPP”
   $ db2 "SELECT COUNT(*) FROM PS_JRNL_LN”
   ```

Appendix A

DB2 UDB Configuration Parameter and Environment Variable Settings Used

a. DB2 UDB Environment Variables

[E]	DB2DBDFT=GLNETAPP
[I]	DB2_EVENT_LOG_CONFIG=NO
[I]	DB2_MIN_DEC_DIV_6=YES
[I]	DB2_SELECTIVITY=ON
[I]	DB2_ANTIJOIN=YES
[I]	DB2_STRIPED_CONTAINERS=ON
[I]	DB2_CORRELATED_PREDICATES=YES
[I]	DB2_HASH_JOIN=YES
[I]	DB2_MMAP_WRITE=OFF
[I]	DB2_MMAP_READ=OFF

Note: [E] indicates environment-level variables that were set at the *.profile* level using the AIX export command, [I] indicates DB2 instance-level variables that were set using the DB2 system command db2set –i.

b. DB2 Database Manager Configuration

```
     Node type = Enterprise Server Edition with local and remote clients

 Database manager configuration release level            = 0x0a00

 CPU speed (millisec/instruction)             (CPUSPEED) = 5.668131e-07
 Communications bandwidth (MB/sec)      (COMM_BANDWIDTH) = 1.000000e+02

 Max number of concurrently active databases     (NUMDB) = 8
 Data Links support                          (DATALINKS) = NO
 Federated Database System Support           (FEDERATED) = NO
 Transaction processor monitor name        (TP_MON_NAME) =

 Default charge-back account           (DFT_ACCOUNT_STR) =

 Java Development Kit installation path       (JDK_PATH) = /usr/java13_64

 Diagnostic error capture level              (DIAGLEVEL) = 3
 Notify Level                              (NOTIFYLEVEL) = 3
 Diagnostic data directory path               (DIAGPATH) =
/home/db2inst1/sqllib/db2dump

 Default database monitor switches
   Buffer pool                         (DFT_MON_BUFPOOL) = OFF
   Lock                                   (DFT_MON_LOCK) = OFF
   Sort                                   (DFT_MON_SORT) = OFF
   Statement                              (DFT_MON_STMT) = OFF
   Table                                 (DFT_MON_TABLE) = OFF
   Timestamp                         (DFT_MON_TIMESTAMP) = ON
   Unit of work                            (DFT_MON_UOW) = OFF
 Monitor health of instance and databases   (HEALTH_MON) = OFF

 SYSADM group name                        (SYSADM_GROUP) = DB2IADM1
 SYSCTRL group name                      (SYSCTRL_GROUP) =
 SYSMAINT group name                    (SYSMAINT_GROUP) =

 Database manager authentication        (AUTHENTICATION) = SERVER
 Cataloging allowed without authority   (CATALOG_NOAUTH) = NO
 Trust all clients                      (TRUST_ALLCLNTS) = YES
 Trusted client authentication          (TRUST_CLNTAUTH) = CLIENT
 Bypass federated authentication            (FED_NOAUTH) = NO

 Default database path                       (DFTDBPATH) = /home/db2inst1

 Database monitor heap size (4KB)          (MON_HEAP_SZ) = 90
 Java Virtual Machine heap size (4KB)     (JAVA_HEAP_SZ) = 1024
 Audit buffer size (4KB)                  (AUDIT_BUF_SZ) = 0
 Size of instance shared memory (4KB)  (INSTANCE_MEMORY) = AUTOMATIC
 Backup buffer default size (4KB)            (BACKBUFSZ) = 1024
 Restore buffer default size (4KB)           (RESTBUFSZ) = 1024

 Sort heap threshold (4KB)                  (SHEAPTHRES) = 61200

 Directory cache support                     (DIR_CACHE) = YES

 Application support layer heap size (4KB)   (ASLHEAPSZ) = 15
 Max requester I/O block size (bytes)         (RQRIOBLK) = 32767
 Query heap size (4KB)                   (QUERY_HEAP_SZ) = 1000
 DRDA services heap size (4KB)            (DRDA_HEAP_SZ) = 128

 Workload impact by throttled utilities(UTIL_IMPACT_LIM) = 100

 Priority of agents                           (AGENTPRI) = SYSTEM
 Max number of existing agents               (MAXAGENTS) = 400
 Agent pool size                        (NUM_POOLAGENTS) = 400
 Initial number of agents in pool       (NUM_INITAGENTS) = 0
 Max number of coordinating agents     (MAX_COORDAGENTS) = (MAXAGENTS -
NUM_INITAGENTS)
 Max no. of concurrent coordinating agents  (MAXCAGENTS) = MAX_COORDAGENTS
 Max number of client connections      (MAX_CONNECTIONS) = MAX_COORDAGENTS

 Keep fenced process                        (KEEPFENCED) = YES
 Number of pooled fenced processes         (FENCED_POOL) = MAX_COORDAGENTS
 Initialize fenced process with JVM     (INITFENCED_JVM) = NO
 Initial number of fenced processes     (NUM_INITFENCED) = 0

 Index re-creation time                       (INDEXREC) = RESTART

 Transaction manager database name         (TM_DATABASE) = 1ST_CONN
 Transaction resync interval (sec)     (RESYNC_INTERVAL) = 180

 SPM name                                     (SPM_NAME) =
 SPM log size                          (SPM_LOG_FILE_SZ) = 256
 SPM resync agent limit                 (SPM_MAX_RESYNC) = 20
 SPM log path                             (SPM_LOG_PATH) =

 TCP/IP Service name                          (SVCENAME) = 60004
 Discovery mode                               (DISCOVER) = SEARCH
 Discover server instance                (DISCOVER_INST) = ENABLE

 Maximum query degree of parallelism   (MAX_QUERYDEGREE) = 1
```

```
Enable intra-partition parallelism     (INTRA_PARALLEL) = NO

No. of int. communication buffers(4KB)(FCM_NUM_BUFFERS) = 4096
Node connection elapse time (sec)         (CONN_ELAPSE) = 10
Max number of node connection retries (MAX_CONNRETRIES) = 5
Max time difference between nodes (min) (MAX_TIME_DIFF) = 60

db2start/db2stop timeout (min)        (START_STOP_TIME) = 10
```

c. DB2 Database Configuration for the GLNETAPP Database

```
Database configuration release level                    = 0x0a00
Database release level                                  = 0x0a00

Database territory                                      = US
Database code page                                      = 819
Database code set                                       = ISO8859-1
Database country/region code                            = 1

Dynamic SQL Query management           (DYN_QUERY_MGMT) = DISABLE

Discovery support for this database       (DISCOVER_DB) = ENABLE

Default query optimization class         (DFT_QUERYOPT) = 5
Degree of parallelism                      (DFT_DEGREE) = 1
Continue upon arithmetic exceptions   (DFT_SQLMATHWARN) = NO
Default refresh age                   (DFT_REFRESH_AGE) = 0
Number of frequent values retained     (NUM_FREQVALUES) = 10
Number of quantiles retained            (NUM_QUANTILES) = 20

Backup pending                                          = NO

Database is consistent                                  = NO
Rollforward pending                                     = NO
Restore pending                                         = NO

Multi-page file allocation enabled                      = NO

Log retain for recovery status                          = NO
User exit for logging status                            = NO

Data Links Token Expiry Interval (sec)      (DL_EXPINT) = 60
Data Links Write Token Init Expiry Intvl(DL_WT_IEXPINT) = 60
Data Links Number of Copies             (DL_NUM_COPIES) = 1
Data Links Time after Drop (days)        (DL_TIME_DROP) = 1
Data Links Token in Uppercase                (DL_UPPER) = NO
Data Links Token Algorithm                   (DL_TOKEN) = MAC0

Database heap (4KB)                            (DBHEAP) = 8686
Size of database shared memory (4KB)  (DATABASE_MEMORY) = AUTOMATIC
Catalog cache size (4KB)              (CATALOGCACHE_SZ) = 1075
Log buffer size (4KB)                        (LOGBUFSZ) = 143
Utilities heap size (4KB)                (UTIL_HEAP_SZ) = 215112
Buffer pool size (pages)                     (BUFFPAGE) = 300000
Extended storage segments size (4KB)    (ESTORE_SEG_SZ) = 16000
Number of extended storage segments   (NUM_ESTORE_SEGS) = 0
Max storage for lock list (4KB)              (LOCKLIST) = 31700

Max size of appl. group mem set (4KB) (APPGROUP_MEM_SZ) = 13480
Percent of mem for appl. group heap   (GROUPHEAP_RATIO) = 70
Max appl. control heap size (4KB)     (APP_CTL_HEAP_SZ) = 128
```

```
Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR) = (SHEAPTHRES)
Sort list heap (4KB)                         (SORTHEAP) = 1912
SQL statement heap (4KB)                     (STMTHEAP) = 4096
Default application heap (4KB)             (APPLHEAPSZ) = 256
Package cache size (4KB)                   (PCKCACHESZ) = 859
Statistics heap size (4KB)               (STAT_HEAP_SZ) = 4384

Interval for checking deadlock (ms)         (DLCHKTIME) = 10000
Percent. of lock lists per application       (MAXLOCKS) = 60
Lock timeout (sec)                        (LOCKTIMEOUT) = -1

Changed pages threshold                (CHNGPGS_THRESH) = 60
Number of asynchronous page cleaners   (NUM_IOCLEANERS) = 3
Number of I/O servers                   (NUM_IOSERVERS) = 8
Index sort flag                             (INDEXSORT) = YES
Sequential detect flag                      (SEQDETECT) = YES
Default prefetch size (pages)         (DFT_PREFETCH_SZ) = 32

Track modified pages                         (TRACKMOD) = OFF

Default number of containers                            = 1
Default tablespace extentsize (pages)   (DFT_EXTENT_SZ) = 32

Max number of active applications            (MAXAPPLS) = 40
Average number of active applications       (AVG_APPLS) = 1
Max DB files open per application            (MAXFILOP) = 64

Log file size (4KB)                         (LOGFILSIZ) = 4096
Number of primary log files                (LOGPRIMARY) = 48
Number of secondary log files               (LOGSECOND) = 0
Changed path to log files                  (NEWLOGPATH) =
Path to log files                                       = /db2logs/NODE0000/
Overflow log path                     (OVERFLOWLOGPATH) =
Mirror log path                         (MIRRORLOGPATH) =
First active log file                                   =
Block log on disk full                (BLK_LOG_DSK_FUL) = NO
Percent of max active log space by transaction(MAX_LOG) = 0
Num. of active log files for 1 active UOW(NUM_LOG_SPAN) = 0

Group commit count                          (MINCOMMIT) = 3
Percent log file reclaimed before soft chckpt (SOFTMAX) = 1920
Log retain for recovery enabled             (LOGRETAIN) = OFF
User exit for logging enabled                (USEREXIT) = OFF

Auto restart enabled                      (AUTORESTART) = ON
Index re-creation time                       (INDEXREC) = SYSTEM (RESTART)
Default number of loadrec sessions    (DFT_LOADREC_SES) = 1
Number of database backups to retain   (NUM_DB_BACKUPS) = 12
Recovery history retention (days)     (REC_HIS_RETENTN) = 366

TSM management class                    (TSM_MGMTCLASS) =
TSM node name                            (TSM_NODENAME) =
TSM owner                                   (TSM_OWNER) =
TSM password                             (TSM_PASSWORD) =
```

Appendix B

a. Sample Application that calls Configuration Advisor

```
/**********************************************************************
** Source File Name = autoconf.sqc
**
** Modified sample program from v72 for Priti Desai
** Usage:
** Start DB2 (because makes a connection!!)
** 1. autoconf <dbname> <userid> <passwd>
**
** Uses CLI to connect and disconnect from db.
**
** Features:
**    - Calls configuration advisor API setting all values by hand
** - if db does not exist or if db2 not started, can trap (not a very
graceful
**    exit)
** - no eee support (you can just use db2_all and the command line
versions)
** - very little testing done
**********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <string.h>
#include <sql.h>
#include <sqlutil.h>
#include <sqlcli1.h>
#include "sqlenv.h"
#include "db2AuCfg.h"

/* Number of input arguments */
int numArgs = 4;


/* Values from by Priti Desai for PeopleSoft Batch, adjust as needed
for your use*/
int version = db2Version810;
int productID = DB2_SG_PID_PS_BATCH;
db2int32 numInputParams = 10; /* Number of input values*/

db2int32 memoryPercentage = 25;
db2int32 workload = DB2_SG_WORKLOAD_MIXED;
db2int32 averageStatement = 1000;
db2int32 transactionsPerMinute = 1000;
db2int32 adminPriority = DB2_SG_ADMIN_PRIORITY_TRANSACTION;
db2int32 isPopulated = DB2_SG_IS_POPULATED_YES;
db2int32 localApplication = 2;
db2int32 remoteApplication = 14;
db2int32 isolationLevel = DB2_SG_ISOLATION_LEVEL_CURSOR_STABILITY;
db2int32 bpResizeable = DB2_SG_BP_RESIZEABLE_YES;
db2int32 apply = DB2_SG_APPLY_DB;
```

```
/***********************************************************
  This procedure prints error information released by the APIs.
***********************************************************/
void printerr(struct sqlca *ca)
{
  char err[512];
  printf("________________ERROR INFO_________________________\n");
  sqlaintp(err, sizeof(err), 80, ca);
  printf("%s\n", err);
  printf("SQLCODE   : %d \n",ca->sqlcode);
  printf("SQLERRMC : %s \n",ca->sqlerrmc);
  printf("SQLERRP   : %s \n",ca->sqlerrp);
  printf("_______________END ERROR INFO______________________ \n");
  return;
}

/*************************************************************
main
*************************************************************/

int main(int argc, char *argv[])
{

  /* Local storage for DB name and Password*/
  char dbName[SQL_ALIAS_SZ+1] = "\0";
  char user[20] = "\0";
  char pswd[20] = "\0";

  if (!(argc == numArgs))
  {
    printf("Incorrect number of arguments.\nCall using autoconf
<dbname> <userid> <passwd>\n");
    return 0;
  }
  else
  {
    /* Take values from the command line*/
    strcpy (dbName, argv[1]);
    strcpy (user, argv[2]);
    strcpy (pswd, argv[3]);
  }


/***************************************************************/
/* Connection to db */
/***************************************************************/

  printf("\nDatabase connection\n");
  printf("--------------------\n");;

  /* allocate an environment handle */
  printf("... allocating environment handle\n");;
  SQLRETURN cliRC = SQL_SUCCESS;
  SQLHANDLE henv; /* environment handle */
  SQLHANDLE hdbc; /* connection handle */
  cliRC = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
  if (cliRC != SQL_SUCCESS)
  {
    printf("Error while allocating the environment handle.\n");
    printf("  cliRC = %d\n", cliRC);
    printf("  line  = %d\n", __LINE__);
    printf("  file  = %s\n", __FILE__);
    return 1;
  }

  /* connect to a database with SQLConnect, */
  /* this is the basic connection */
  /* allocate a database connection handle */
  printf( "... allocating connection handle\n");
  cliRC = SQLAllocHandle( SQL_HANDLE_DBC, henv, &hdbc ) ;
  if (cliRC < 0)
  {
    printf("Error allocating connection environment handle\n");
    goto envexit;
  }

  /* connect to the database */
  printf ("... connecting to the database %s \n", dbName);
  cliRC = SQLConnect( hdbc,
                    (SQLCHAR *)dbName, SQL_NTS,
                    (SQLCHAR *)user, SQL_NTS,
                    (SQLCHAR *)pswd, SQL_NTS
                  ) ;
  if (cliRC < 0)
  {
    printf("Error connecting to database\n");
    goto exit;
  }
  printf("... done connection steps\n");


/********************************************************************/
/* set up the interface to the api */
/*********************************************¡**********************/
  db2AutoConfigInterface autoConfigInterface;
  autoConfigInterface.iParams.pElements = (db2AutoConfigElement *)
malloc(sizeof(db2AutoConfigElement)*numInputParams);
  autoConfigInterface.iProductID = productID;
  strcpy(autoConfigInterface.iDbAlias, dbName);

  strcpy(autoConfigInterface.iProductVersion,
DB2_SG_PRODUCT_VERSION_1_1);

  autoConfigInterface.iApply = apply;
  autoConfigInterface.iParams.numElements = numInputParams;
  autoConfigInterface.iParams.pElements[0].token =
DB2_SG_MEMORY_PERCENTAGE;
  autoConfigInterface.iParams.pElements[0].value = memoryPercentage;
  autoConfigInterface.iParams.pElements[1].token = DB2_SG_WORKLOAD;
  autoConfigInterface.iParams.pElements[1].value = workload;
  autoConfigInterface.iParams.pElements[2].token  =  DB2_SG_NUM_STATEMENT;
  autoConfigInterface.iParams.pElements[2].value = averageStatement;
  autoConfigInterface.iParams.pElements[3].token =
DB2_SG_TRANS_PER_MINUTE;
  autoConfigInterface.iParams.pElements[3].value =
transactionsPerMinute;
  autoConfigInterface.iParams.pElements[4].token =
DB2_SG_ADMIN_PRIORITY;
  autoConfigInterface.iParams.pElements[4].value = adminPriority;
  autoConfigInterface.iParams.pElements[5].token = DB2_SG_IS_POPULATED;
  autoConfigInterface.iParams.pElements[5].value = isPopulated;
  autoConfigInterface.iParams.pElements[6].token =
DB2_SG_LOCAL_APPLICATION;
  autoConfigInterface.iParams.pElements[6].value = localApplication;
  autoConfigInterface.iParams.pElements[7].token =
DB2_SG_REMOTE_APPLICATION;
  autoConfigInterface.iParams.pElements[7].value = remoteApplication;
  autoConfigInterface.iParams.pElements[8].token =
DB2_SG_ISOLATION_LEVEL;
  autoConfigInterface.iParams.pElements[8].value = isolationLevel;
  autoConfigInterface.iParams.pElements[9].token  =  DB2_SG_BP_RESIZEABLE;
  autoConfigInterface.iParams.pElements[9].value = bpResizeable;



/*******************************************************************/
/* call the api */

/*******************************************************************/
  SQL_API_RC rc;
  struct sqlca sqlca;
  printf("\nRunning the wizard\n");
  printf("------------------\n");
  printf("... calling the api\n");
  rc = db2AutoConfig(version, &autoConfigInterface, &sqlca);
  // print results
  if (rc == DB2_SG_RC_OK)
  {
    /* must free all the memory allocated by db2AutoConfig() */
    db2AutoConfigFreeMemory(version, &autoConfigInterface, &sqlca);
  }
  else
  {
     printerr(&sqlca);
  }
  printf("... done wizard steps\n");

/******************************************************************/
/* Disconnect */
/******************************************************************/
  printf("\nDatabase disconnect\n");
  printf("---------------------\n");

  /* disconnect from the database */
  printf("... disconnecting from db\n");
  {
  int rc = SQLDisconnect( hdbc ) ;
  if (rc < 0) printf("Error disconnecting from db\n");
  }
```

```
exit:

  /* free the connection handle */
  printf("... freeing connection handle\n");
  {
  int rc = SQLFreeHandle( SQL_HANDLE_DBC, hdbc ) ;
  if (rc < 0) printf("Error freeing connection handle\n");
  }

envexit:
  /* free the environment handle */
  printf("... freeing environment handle\n");
  {
  int rc = SQLFreeHandle( SQL_HANDLE_ENV,  henv ) ;
  if (rc < 0) printf("Error freeing environment handle\n");
  }

  return 0;
}
```

b. Sample Application Makefile

```
# file modified from samples makefile

#########################################################################
#                  1 -- COMPILERS + VARIABLES
#########################################################################

DB2PATH = $(HOME)/sqllib


# Use the compiler
CC= xlC
# Use xlc_r for multi-threaded programs
#CM=xlc_r
# For multi-threaded programs on AIX 4.3 or later, use:
# CM=xlc_r7
# For 64 bit executables use:
#EXTRA_CFLAGS=-q64

# The required compiler flags
CFLAGS= $(EXTRA_CFLAGS) -I$(DB2PATH)/include

# The required libraries
LIBS= -L$(DB2PATH)/lib -ldb2
LIBS1= -L$(DB2PATH)/lib -ldb2 -ldb2apie

autocfg : autocfg.C
      $(CC) -o autocfg autocfg.C $(CFLAGS) $(LIBS)
```

References

Further information about the products used in the study is available at:

DB2 Universal Database:	http://www-4.ibm.com/software/data/
Network Appliance Enterprise Servers:	http://www.netapp.com/products/filer