106

AIX update

August 2004

In this issue

Published by

Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

Phone: 214-340-5690
Fax: 214-341-7081

Editor

Trevor Eddolls
E-mail: trevore@xephon.com

Publisher

Nicole Thomas
E-mail: nicole@xephon.com

Subscriptions and back-issues

A year's subscription to *AIX Update*, comprising twelve monthly issues, costs $275.00 in the USA and Canada; £180.00 in the UK; £186.00 in Europe; £192.00 in Australasia and Japan; and £190.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the November 2000 issue, are available separately to subscribers for $24.00 (£16.00) each including postage.

AIX Update on-line

Code from *AIX Update*, and complete issues in Acrobat PDF format, can be downloaded from our Web site at http://www.xephon.com/aix; you will need to supply a word from the printed issue.

Disclaimer

Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, scripts, and other contents of this journal before making any use of it.

Contributions

When Xephon is given copyright, articles published in *AIX Update* are paid for at the rate of $160 (£100 outside North America) per 1000 words and $80 (£50) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of $32 (£20) per 100 lines. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

Printed in England.

Using sudo

Wouldn't it be a good idea to allow certain users to run only certain privilege commandswithout their needing to know the password of the account they will run under to run that command? Well, by using **sudo** (super-user-do) you can. **Sudo** can be found on the AIX Linux toolbox CD or downloaded from the Bull Web site (current version 1.6.3.6).

Sudo allows the partitioning of root (or another user's) privileges; it also provides good logging to the system logs for auditing purposes. **Sudo** can be used on a per host basis (as in most cases) or throughout the network. Here's how **sudo** works. The file */etc/sudoers* is the main configuration file; the *sudoers* file informs **sudo** which user can run which commands on what host, and whom they can masquerade as for the duration of the command execution. After executing **sudo** the user is prompted for their own password, and not the password of the user account they wish to change to. This is one of the big differences between **su** and **sudo** and it means that passwords are not exchanged between users.

Once the user has supplied their password to **sudo**, **sudo** will not ask for authentication again when running **sudo**-based commands during the next five minutes. After that time the user will be prompted for their password again when they issue a **sudo** command.

If an unauthorized person tries to use **sudo**, mail is automatically sent to root and the event is logged, generally to */var/adm/messages*.

THE SUDOERS FILE

The most basic format of the *sudoers* file is:

```
user host  =  (user-to-alias) command-to-run
```

By default the *sudoers* file is installed as:

```
root ALL=(ALL) ALL
```

which translates to: give user root (root) **sudo** access to every host (ALL) as any user (ALL) running any command (ALL).

A more typical scenario would be when a user covers for the systems administrator when that person is away. This type of user would need complete root access, as the example below shows:

```
pjohn, klyn uksys01 =(ALL) ALL
```

users *pjohn* and *klyn* have **sudo** access to the host *uksys01* as any user running any command.

The real power of **sudo** is the ability to use alias entries in the *sudoers* file and its logging capabilities. These aliases are defined by the entries User_Alias, Host_Alias, Runas_Alias, and Cmnd_alias. Looking at a rather simple, *sudoers* entry, suppose we had two users, dxtan and lpete, who have been authorized to stop/start the mail server and also to bounce the local machine:

```
User_Alias    ADMINS=dxtan, lpete

Cmnd_Alias    REBOOT=/usr/sbin/shutdown
ADMINS ALL= (ALL) REBOOT
jpaul sysuk01 = (root) NOPASSWD: /usr/sbin/stopsrc -s sendmail
jpaul sysuk01 = (root) NOPASSWD: /usr/sbin/startsrc -s 'sendmail' -a '-
db -q30m'
```

In the above example under the User_Alias, we define the users and the alias ADMINS expands to two user IDs, namely *dxtan* and *lpete*.

In the Cmnd_Alias entry we define the commands to be run, and the aliases REBOOT expands to the shutdown command. Next we tell **sudo** what commands can be run by whom; in this case the entry ADMINS tells **sudo** that ALL members of ADMIN (dxtan and lpete) can run the commands contained in the alias's REBOOT. In short we are stating that users dxtan and lpete can run only the command **/usr/bin/shutdown** and no other commands. That means they cannot run shutdown with other

parameters, like **shutdown –m +2**. This keeps authentication and security issues intact. User *jpaul* on host *sysuk01* can alias as root and run the service commands that stop and start sendmail. When evoking either the stopsrc or startsrc with the commands listed in *sudoers*, user jpaul will not (*NOPASSWD*) be prompted for his password, unlike users *dxtans* and *lpete*, who will be prompted for their password when executing the shutdown command.

Be advised that the use of the *NOPASSWD:* option should be given only to users who are running batch scripts or remote jobs. It is not really a good idea to allow users to run commands interactively with no password prompt, even if governed by **sudo**.

Sudo is well equipped to be run over a network system, where one can centrally control access from one file. Suppose we had the same users again, dxtan and lpete, who have been authorized to maintain the DNS servers, using the **named** and **rndc** command. User lpete is allowed to maintain only the primary and secondary DNS servers on the hosts sysuk02.myhost.co.uk and sysuk03.myhost.co.uk, whilst user dxtan is allowed to maintain only the forwarding DNS at uksys04.myhost.com. The following example accomplishes this scenario:

```
Host_Alias DNSMAIN = sysuk02.myhost.co.uk, sysuk03.myhost.co.uk
Host_Alias DNSFORW =    sysuk04.myhost.co.uk
User_Alias      DNSUSER = lpete
User_Alias      DNS2USER = dxtan
Runas_Alias     ADMIN = root
Cmnd_Alias NAMED = /usr/sbin/named, /usr/sbin/rndc
DNSUSER  DNSMAIN = NAMED
DNS2USER  DNSFORW = NAMED
```

Sudo allows the use of group IDs instead of specifying single users, as in the */etc/group* file. Imagine you have a number of users who belong to the group ‘admins’. You can include all these users without hardcoding their names in the User_Alias. Just use the % sign like so:

```
%admins ALL=(ALL) ALL
```

Sudo will pull these group members from the */etc/group* file.

So far we have looked at aliasing to root, but, as mentioned previously, root is not the only privilege user. Imagine we had a couple of users (dxtan and lpete) who needed to **sudo** to the user ID sybase to start the database servers on the hosts sysuk05 and sysuk06. The sybase account is generally the account used to install sybase dataservers. In the following example the asterisk used in the Cmnd_Alias indicates that the user can parse any valid command parameter to the command startserver.

```
Cmnd_Alias SYBASESRV =/opt/sybase/install/startserver *

dxtan,lpeter  sysuk05.mycompany.co.uk = (sybase) SYBASESRV
```

RUNNING SUDO

Once **sudo** has been set up, users can determine what commands they can run using **sudo –l**:

```
$ sudo -l
```

User lpete may run the following commands on this host:

```
(root) /usr/sbin/shutdown
(root) /usr/sbin/stopsrc -s sendmail,
(root) /usr/sbin/startsrc -s 'sendmail' -a '-db -q30m'
(sybase) /opt/Sybase/install/startserver *
```

Running a **sudo** command is straightforward. To run the command for which you have been authorized in the *sudoers* file, all you need to do is prefix the command with **sudo**. You are then prompted for your own password before execution of the command or process is initiated (assuming the NOPASSWD has not been specified for that command in the *sudoers* file):

```
$ sudo -u dxtan '/usr/sbin/named restart'
      We trust you have received the usual lecture from the local System
      Administrator. It usually boils down to these two things:
        #1) Respect the privacy of others.
        #2) Think before you type.

      Password:
```

If the NOPASSWD has been used in *sudoers*, no password prompt will appear. Alternatively, one can specify the username

using the **-u** option. This is used if running a command other than root, though as a general rule I always specify the **-u**, even if I am running a non-root-based command. That way I know **sudo** will pick up the correct entry – the one that I am allowed to use.

If an unauthorized user tries to run a command, **sudo** will inform the user and then mail root promptly as well as log the event to the messages file.

```
$ sudo -u dxtan '/usr/bin/topas'
Sorry, user dxtan is not allowed to execute '/usr/bin/topas'
```

PARSING ARGUMENTS TO SUDO

It is inevitably the case that some scripts will need **sudo** to run some commands – the most notable ones are application-specific user IDs that may need to run a command as root. Suppose during script execution it needs to delete some files, but it does not have the right privileges to do so. By parsing variables into **sudo**, these will get expanded. The only rule to note is that, once expanded, the command line must match exactly what is in the *sudoers* file. For example, suppose a script command is required to delete a directory as part of a batch script execution, where the directory name was parsed to the script thus:

```
  sudo -u root /usr/bin/rm -r /home/db2inst1/sqllib/function/routine/
sqlproc/${DIR_NAME}
```

In the *sudoers* file you would need to insert that expanded variable name. It may be the case that several entries would be required to cover all known variable expanded values contained in the script, like in the example above for {DIR_NAME}:

```
db2admin  uksysØ1 = (root) NOPASSWD: /usr/bin/rm -r /home/db2inst1/
sqllib/function/routine/sqlproc/SAMPLE1

db2admin  uksysØ1 = (root) NOPASSWD: /usr/bin/rm -r /home/db2inst1/
sqllib/function/routine/sqlproc/SAMPLE2
```

The above example is one method that allows the successful execution of shell-based commands from within a script, without

the need to know the password of root – this is because it is controlled by **sudo**.

One note of caution: when using **sudo** to execute user-based scripts, if the script shells out to run a system-type command, that command, as well as the actual user script name, should feature in the *sudoers* file. If it does not, the execution could fail. And remember to include the full pathname to the command in *sudoers*!

LOG IT

By default all logging is carried out to syslog using the local2 facility. If left untouched, all messages will go to */var/adm/messages*. I personally do not think this is a good idea. You do not want to be filtering through the main messages file looking for **sudo** violations. I would recommend directing all **sudo** messages to their own log file. To accomplish this simply put the following entry into */etc/syslog.conf*:

```
local2.debug    /var/adm/sudo.log
```

First create the *sudo.log* then restart syslogd:

```
# > /var/adm/sudo.log
# stopsrc -s syslogd
# startsrc -s syslogd
```

Sudo will generate a lot of messages – probably more than you would expect to see – but, because we are dealing with security, in my view it is better to have more than less:

```
May 20 07:23:44 sysuk05 sudo :  dxtan : TTY=pts/5 ; PWD=/home/dxtan;
USER=root ; COMMAND=/opt/Sybase/install/startserver
```

In the above extract from the log file we can tell that user dxtan on 20 May at 07:23 started the Sybase server by running the command */opt/Sybase/install/startserver*. One thing to note here is that **sudo** logs the successful and failed attempts, but does not log events after **sudo** was evoked. For that you will have to look at the application-specific log files or the messages file. For failed **sudo** attempts where a user tried to run a root privilege command that he is not authorized to do, we actually

have proof that the user does indeed have a set UID of 0 (ie root).

```
May 20 07:30:32 sysuk05 sudo: authentication failure; logname=dxtan
uid=0 euid=0 tty=pts/5 ;  ruser= rhost= user dxtan
```

Once your system is up and running with **sudo**, it may make sense to disable the **su** command completely so ordinary users can't use it to change into other users' accounts. If this suits your application/administration needs, simply take away the SETUID from */bin/su* – of course, you have to do this as root. Now if some user tries to **su** to say root or another user, even if they know the password, they will not be able to change accounts (although root will still be able to). This approach though not totally foolproof will stop most users from exploiting the security loophole of gaining root access. If a user knows the root password then he or she can either rlogin/rsh, assuming that the 'r*' commands are not commented out in */etc/inetd.conf*, or they can create a new login shell to gain access if they know the root password. So make sure your security policy ensures regular changing of all root passwords.

A LITTLE HELP WITH EDITING THE SUDOERS FILE

When editing the *sudoers* file use **visudo**, usually located in */usr/local/bin*. This **sudo**-assisted utility enables the user to lock the file whilst in editing mode, and it also does basic syntax checking when saving the file. It will prompt you to see whether you wish to re-edit the file if errors are found; it will also display the error prone line numbers.

Visudo will not do any sanity checks of processes, command names or host names, it just checks the syntax of the configuration file. To use your favourite editor for this, simply make sure it has been assigned to the EDITOR environment variable and has been exported:

```
# export EDITOR=vi
Now run visudo to edit the sudoers file:
# visudo
```

CONCLUSION

Sudo is a good user security access utility, which should be the preferred choice instead of using the **su** command. **Sudo** allows you practically unlimited control over which users can run commands under root or any other privilege user.

David Tansley
Global Production Support (IBM p-series) (UK)

AIX system cloning AIX 4.3.3 – using the NIM environment

The May 2004 issue of *AIX Update* contained the first part of this article, which looked at cloning using tape (see *AIX system cloning AIX 4.3.3* in issue 103).

This month we look at cloning using the NIM environment.

USING THE NIM ENVIRONMENT

First of all you need a mksysb resource from the source (rsx_master) system.

Roughly estimate the size of the mksysb image that will be created at the source system. You can do this with this code sample (it excludes parts that will not be backed up):

```
lsvg -l rootvg \
| egrep -v '(paging|jfslog|boot|closed|rootvg:|LV NAME|N/A)' \
| while read hd type lp pp pv lvstate mnt
do
df -k $mnt | tail -1 | awk '{print $2-$3}'
done | awk '{ s=s+$1 } END { print s " KB" }'
```

Export a part of the NIM master filesystem where this amount of space will be available.

At the nim_master system:

```
smit mknfsexp
```

or from the command line:

```
/usr/sbin/mknfsexp -d '/nim/rsx_clone/bz' -t 'rw' -c 'rsx_master' -r
'rsx_master' '- B'
```

At the source system rsx_master mount the filesystem:

```
smit mknfsmnt
/usr/sbin/mknfsmnt -f '/bck' -d '/nim/rsx_clone/bz' -h
                                                    'nim_master' '-n' \
'-N' '-a' -t 'rw' -w 'bg' '-Y' '-Z' '-X' '-H' '-j' '-q' '-g'
ulimit -f unlimited # to allow root to create files greater 2GB
```

Start the back-up to the nfs part of the nim_master:

```
/usr/bin/mksysb '-i' /bck/master.bin
```

Wait for the end of the system back-up, then:

```
umount /bck
```

Go to the nim_master system – you can use the custom.ksh script, the bosinst.data, and the image.data file from the first step (see pages 23–31 of issue 103).

Put them in a directory, eg */nim/rsx_clone/ores*.

Define the system that will receive the system back-up, rsx_clone, as a NIM client.

The node name, rsx_clone, must already resolve to a defined IP address.

The Network Adapter Hardware Address is a prerequiste for the */etc/bootptab* entry for the client.

```
smit nim_mkmac

                              Define a Machine

                                                    [Entry Fields]
* NIM Machine Name                                  [rsx_clone]
* Machine Type                                      [standalone]          +
* Hardware Platform Type                            [chrp]                +
 Kernel to use for Network Boot                     [up]                  +
 Primary Network Install Interface
*   Cable Type                                      tp                    +
*   NIM Network                                     nw192_230
```

```
*   Host Name                                       rsx_clone
   Network Adapter Hardware Address                [0004ac9756a6]
   Network Adapter Logical Device Name             [ent0]
```

Adjust a few entries in the bosinst.data file. Set the PROMPT to no and the CUSTOMIZATION_FILE entry empty (because we do not plan to enter anything at the install screens and we are not using floppy disks, we supply the customization information through the NIM script resource).

You can make any customization changes you like to the script, but you cannot make changes to the network interface. This is because the script is located on the nim_master and is NFS mounted when it is executed. A change to the IP address can be achieved through the install option:

```
Remain NIM client after install? [yes]
```

If you like to change additional network interfaces, the safest way is to append all this to the */etc/firstboot* file. This is because the file is executed after the installation/restore at the end of the first reboot.

SAMPLE CONTENT OF CUSTOM.KSH

```
#!/usr/bin/ksh
# first make all the changes that can be done without network
#  interface changes
#
mv /etc/sendmail.cf /etc/_sendmail.cf
# prevent sending of old mail
#
# now prepare the changes that will be done at the end of the first
reboot
cat >> /etc/firstboot <<EOF
#
# put in all changes where IP address will change
#
EOF
```

Create NIM resources with the following types:

- */nim/rsx_clone/bz/master.bin* – mksysb resource
- */nim/rsx_clone/ores/custom.ksh* – script resource

- */nim/rsx_clone/ores/bosinst.data* – bosinst.data resource
- */nim/rsx_clone/ores/image.data* – image.data resource.

smit nim_mkres

Your NIM system administrator has to prepare a spot and lpp resource at the NIM master that matches the AIX level from the rsx_master system.

The following NIM resources should be available after that step:

```
lsnim | grep resource
aix433up                resources       lpp_source
spot433up               resources       spot
rsx_clone_mksysb        resources       mksysb
rsx_clone_script        resources       script
rsx_clone_bosinst       resources       bosinst_data
rsx_clone_image         resources       image_data
```

Define the install/restore action. This step combines the NIM resources created and defines the restore activity.

```
smit ...
Install the Base Operating System on Standalone Clients
Type or select values in entry fields.
Press Enter AFTER making all desired changes.
[TOP] [Entry Fields]
* Installation Target rsx_clone
* Installation TYPE mksysb
* SPOT spot433up
* LPP_SOURCE aix433up
MKSYSB rsx_clone_mksysb
BOSINST_DATA to use during installation [rsx_clone_bosinst] +
IMAGE_DATA to use during installation [rsx_clone_image] +
RESOLV_CONF to use for network configuration [] +
Customization SCRIPT to run after installation [rsx_clone_script] +
Remain NIM client after install? [yes] +
PRESERVE NIM definitions for resources on [yes] +
FORCE PUSH the installation? [no] +
Initiate reboot and installation now? [yes] +
-OR-
Set bootlist for installation at the [no] +
next reboot?
```

AFTER THIS STEP

After this step check:

1 Whether the */etc/tftpboot* file contains an entry for the rsx_master system:

```
cat /etc/tftpboot | grep rsx_clone
rsx_clone:bf=/tftpboot/
rsx_clone:ip=ccc.ccc.ccc.ccc:ht=ethernet:ha=HHHHHHHHHHHH:\
sa=sss.sss.sss.sss:gw=ggg.ggg.ggg.ggg:sm=mmm.mmm.mmm.mmm
```

where:

– *ip* is the ip address of rsx_clone.

– ha is the hardware mac address of the network card

– sa is the address of the bootserver

– gw is the default gateway of rsx_clone

– sm is the subnetmask of rsx_clone

– bf is the boot image for rsx_clone, which will be downloaded by the next network boot.

2 That all the allocated NIM resources have been exported to the rsx_clone system:

```
exportfs | grep rsx_clone
/nim/rsx_clone/ores/custom.ksh  -ro,root=rsx_clone:,access=rsx_clone:
/nim/rsx_clone/ores/bosinst.data  -ro,root=rsx_clone:,access=rsx_clone:
/nim/rsx_clone/ores/image.data  -ro,root=rsx_clone:,access=rsx_clone:
/nim/rsx_clone/bz/master.bin  -ro,root=rsx_clone:,access=rsx_clone:
```

Reboot the rsx_clone system; go to the system console.

On PCI systems, enter 1 on the serial console or F1 on the graphic console.

When the keyboard symbol shows up on the screen:

```
RS/6000 Firmware
Version TCP03126
(c) Copyright IBM Corp. 2000 All rights reserved.
-------------------------------------------------------------------------
Utilities

1 Set Passwords and Unattended Start Mode
2 SCSI Spin Up
3 Display Error Log
4 Remote Initial Program Load Setup
```

```
5 Change SCSI Id
6 Update System Firmware
7 Select Console

--------------------------------------------------------------------------
Network Parameters

1. IP Parameters
2. Adapter Parameters
3. Ping

RS/6000 Firmware
Version TCP03126
(c) Copyright IBM Corp. 2000 All rights reserved.
--------------------------------------------------------------------------
IP Parameters

1. Client IP Address [ccc.ccc.ccc.ccc]
2. Server IP Address [sss.sss.sss.sss] <-- the ip of the nim server
3. Gateway IP Address [ggg.ggg.ggg.ggg]
4. Subnet Mask [mmm.mmm.mmm.mmm]
```

Define the bootdevice to be the network interface (ent0). Execute a ping from the menu. If this works OK, exiting the menu will start the network boot.

You will see the bootp sequence – after that the automatic system restore starts.

At the end, the system reboots and the configuration of the customization script are taken.

Now it is time to clean up old configurations (eg sendmail.cf, mailqueue, etc) and adjust to the new hostname.

Imhotep
Unix Systems Administrator (Austria)

Send your article for *AIX Update* to the editor, Trevor Eddolls, at TrevorE@xephon.com. A copy of our Notes for Contributors can be downloaded from www.xephon.com/nfc.

Timeout-enabled interactive input

INTRODUCTION

When writing a shell script that will accept input interactively, you may wish to build in an ability for the script to time out. This requires an understanding of signal handling.

When a read command is issued, it will wait for an indefinite period for an input or until the process is killed and, therefore, nothing can be done to intercept this from the script that issued the read command. A second process is required as a time keeper, which will, at the end of a specified period, send a signal to the input-accepting script to signal the timeout. The input-accepting script then captures that signal and processes it the way it sees fit.

TOEII.KSH

```
################################################################################
#  Name       : toeii.ksh (Time Out Enabled Interactive Input)
#  Overview : The script demonstrates how to implement a timeout
#                    functionality while accepting
#                    an input interactively.
#
#  Notes    : 1. The script contains the following functions:
#                       o main
#                       o InitialiseVariables
#                       o PrepareTimerScript
#                       o HandleTimeOutSignal
#                       o AcceptInput
################################################################################
#   Name     :  InitialiseVariables
#   Overview : The function initializes the required variables
#   Notes    :
################################################################################
InitialiseVariables ()
{
#
# process id for this process
#
export THIS_PROCESS_PID=$$
```

```
#
# script that will run in the background as timer
#
export  TIMER_SCRIPT=/tmp/ts_${THIS_PROCESS_PID}.ksh
#
# signal that will be sent to this process by the process that will run
# the timer script
#
export TIMER_SIGNAL=3
#
# required sleep duration within which an input must be provided;
#  otherwise; a message will be displayed and the
#  script will exit
#
export SLEEP_DURATION=1Ø
}
################################################################################
#   Name     :  PrepareTimerScript
#   Overview : The function prepares a script that will run in
#                   the background as a timer
#   Notes    :
################################################################################
PrepareTimerScript ()
{
cat <<! > ${TIMER_SCRIPT}
sleep ${SLEEP_DURATION}
kill  -${TIMER_SIGNAL} ${THIS_PROCESS_PID} 2> /dev/null
!
chmod 755 ${TIMER_SCRIPT}
}
################################################################################
#   Name     :  HandleTimeOutSignal ()
#   Overview : The function is executed when the signal, $TIMER_SIGNAL
#                   is caught by this process
#   Notes    :
################################################################################
HandleTimeOutSignal ()
{
  echo "Time out for input occurred; Re-run the program"
  #
  # remove the timer script
  #
  rm -f ${TIMER_SCRIPT}
  #
  exit Ø
}
################################################################################
#   Name     :  AcceptInput ()
#   Overview : The function waits for certain period of time to accept
#                     an input. When the expected wait
```

```
#                     duration is expired and no input is supplied,
#                     the script displays a message,
#                     which is displayed by HandleTimerSignal ().
#   Notes    :
################################################################################
AcceptInput ()
{
echo "Enter  Input:\c"
read INPUT
#
# remove timer script
#
rm -f ${TIMER_SCRIPT}
}
#
################################################################################
#   Name     :  main
#   Overview : The function implements the process structure.
#   Notes    :
################################################################################
main ()
{
InitialiseVariables
#
PrepareTimerScript
#
# set the trap to handle incoming signal
#
trap "HandleTimeOutSignal"  3
#
# execute timer script in the background
#
${TIMER_SCRIPT}  &
#
# accept input
#
AcceptInput
}
#
# invoke main
#
main
```

Teach me DB2 on AIX!

Editor's note: this is the first part of a series of articles looking at DB2 UDB running on AIX and comparing it with DB2 on mainframes.

This is a catchy but misleading title because one cannot learn how to administer DB2 on AIX without taking formal educational courses and having at least a bit of hands-on experience in performing database administration in UDB DB2 on AIX.

For ease in writing this article I shall refer to DB2 on OS/390 as DB2 or simply as OS/390, and DB2 on AIX simply as UDB. In both platforms I shall be talking about DB2 Version 8.1 in a non-data sharing environment on OS/390 and in a non-partitioning environment in EE on AIX.

The luxury of a DBA being a single platform DBA is fading quickly. Many DBAs find themselves responsible for supporting various DBMSs, such as DB2, Oracle, and SQL Server, on various platforms, such as OS/390, AIX, Windows, and Linux.

I am a long-time OS/390 DBA recently forced to pay attention to DB2 on the AIX platform. I wrote this article as an aid to myself and, hopefully, to help others like myself.

The best way to compare and contrast major concepts of DB2 on both platforms is to show them in a tabular manner, like on an Excel spreadsheet, which I have for my own personal use. However, because the tabular format in the context of this article is not possible, I chose to display the information in bullet points and elaborate on it in narrative form when needed – even at the expense of repeating the same point under two headings for clarity.

Several sources of information, Red Books, manuals, and articles, were consulted in writing this article. These sources are too numerous to mention individually. I hereby acknowledge all these references and their contribution to my knowledge and

to this article. I alone bear the sole responsibility of any error inadvertently introduced.

INSTALLATION

OS/390:

- SMP/E RECEIVE, APPLY, ACCEPT the DB2 software.
- Run installation CLIST in Install, Migrate, or Update mode to generate JCL jobs.
- Modify JCL jobs and submit them, following a strict sequence of installation or migration steps specified in the installation manual.
- V8.1 supports 64-bit addressability with dramatic impact on virtual storage like buffer pools and sizes of address spaces. Also DB2 V8.1 needs certain levels of hardware.

UDB:

- Log in as root.
- Mount the CD-ROM.
- From the same mount directory key in **db2setup** and follow the installation panels.
- Create two instances: one called DAS, used for the GUI 'stuff', the other a User instance used for applications. The command to create the instance is **$db2icrt**.
- Each instance has three things related to it:
  - user id to be used as instance admin id
  - Cfg file (dbm)
  - AIX id called owner id of the instance.
- The admin id and the instance owner id are regular AIX userids created by an AIX Administrator outside DB2 UDB

jurisdiction. These can be created by the AIX commands **$mkgroup** and **$mkuser**.

- Think of the UDB instance like a DB2 subsystem on the mainframe. One can start it by issuing **$db2start** or stop it by issuing **$db2stop**. One can even list all the instances that somebody else has created on that AIX machine with the command **$db2ilist**.
- One can create several UDB instances on the same AIX machine because they are installed in separate paths.
- Generally AIX DB2 products are installed in */usr/lpp/db2-vv-rr*.
- At a high level, the sequence of installation of UDB on AIX is:
  - install the DB2 software and create the instance on the AIX server using the installation CD (the software comes on CD)
  - once the DB2 UDB software is installed, configure DB2 on the AIX server
  - if one has client WSs that have to connect to the AIX server, one needs to:
    - install the Runtime Client (CAE) on each client from a CD-ROM and, as usual, run **db2setup** and follow the panels.
    - once the Runtime Client (CAE) is installed on each client, the CAE on each client needs to be configured.

SUBSYSTEM VERSUS INSTANCE

OS/390:

- Basically a subsystem is a program or group of programs that run under MVS.

- DB2 is a subsystem.
- The DB2 subsystem is defined to MVS in SYS1.PARMLIB.
- The DB2 subsystem acts as a server of data for requesters of data such as batch jobs, CICS, and other DRDA data requesters.

UDB:

- The closest equivalent to an MVS DB2 subsystem on AIX is the DB2 instance.

ADDRESS SPACES

OS/390:

- The DB2 subsystem has several address spaces – MSTR, DBM1, IRLM, and DDF. Because of the 64-bit addressability, the size of some of these address spaces has increased enormously. For example the size of the DBM1 address space has gone from 2 gigabytes to (2**64) 16 exabytes.
- These address spaces come up to life when the MVS console operator issues the command **-DSNx START DB2**.
- One can see these DB2 address spaces coming up in TSO running as started tasks in IOF or equivalent.

UDB:

- The closest equivalent to an MVS DB2 address space in UDB AIX is a process.
- The most famous UDB process is db2sysc.
- These processes come to life when the instance owner issues the command **$db2start**.
- One can see these DB2 UDB processes running by issuing the command **$ps -f grep(db2)**.

DB2 CATALOG AND DIRECTORY

OS/390:

- Every DB2 subsystem has its own catalog (DSNDB06) and directory (DSNDB01) (unless it is a member of DB2 datasharing group, in which case it shares one catalog with other members of the group). Datasharing is beyond the scope of this article.
- Actually DSNDB06 and DSNDB01 are the names of the databases of the catalog and directory in OS/390.
- The catalog and directory of the DB2 subsystem and the underlying VSAM linear datasets are created from the JCL jobs that the installation CLIST generates.
- The generating CLIST will compute the sizes for the catalog and directory TSs from values entered on the installation panels when running the installation CLIST.

UDB:

- In contrast to DB2 on OS/390, the instance on AIX does not have its own catalog and directory databases.
- UDB has a collection of databases. Each UDB database has its own catalog, its own logs, and its own cfg file.
- The catalog in UDB and its various file structures for a particular UDB database are created when a CREATE DATABASE command is executed. At this time a TS called SYSCATSPACE is automatically computed from default values to house the UDB catalog tables.
- I will talk more about databases below, but here is an example of how to create a UDB database called Nick:

```
CREATE DATABASE  nick ON  '/db2main'
    CATALOG TABLESPACE MANAGED BY SYSTEM USING
                                ('/db2cat001/ind1rada/drad011')
    USER TABLESPACE MANAGED BY SYSTEM USING
                        ('/db2dev001/ind1rada/drad011/userspace1',
                        '/db2dev002/ind1rada/drad011/userspace1' )
    TEMPORARY TABLESPACE MANAGED BY SYSTEM USING
```

```
'/db2dev001/ind1rada/drad011/tempspace1',
'/db2dev002/ind1rada/drad011/tempspace1'' )
```

- Remember that in UDB the CREATE DATABASE is not an SQL statement as is the case in OS/390 DB2. It is a UDB command.

DATABASES, TABLESPACES, AND TABLES

OS/390:

- Generally in OS/390 one creates the database first, independently, then creates the TSs (using either stogroup-defined datasets or user-defined datasets). That causes the underlying linear datasets to be allocated.
- SMS can handle the placement of the linear datasets, but will not manage the space.
- The space allocation values for the linear datasets are taken from the Pri and Sec quantities parameters specified in the TS definition, or in the IDCAMS parameters in the case of user-defined datasets.
- Tables in OS/390 have the same structure as in UDB, ie rows, columns, primary key, foreign key, referential integrity, etc, and can have an index on any of the columns or combination of columns.
- Database administration operations on tables are the same in both platforms, such as creating a table, dropping a table, altering a table, adding a column to a table, etc.
- The indexes, of course, are type 2 indexes.
- There are two types of table in OS/390:
  - system tables, which are called the catalog and directory tables, and have a qualifier of SYSIBM. These are per DB2 subsystem, not per database as in UDB.
  - user tables such as permanent tables and temporary tables.

- In addition there are new concepts that relate to tables such as triggers, row ids, identity columns, MQTs, sequences. These concepts are the same for OS/390 as for UDB.

UDB:

- In UDB the situation is different. Once the **CREATE DATABASE <database name>** command is executed, three TSs of the SMS type are created automatically and dynamically by AIX using default names and sizes. These default TSs are syscatspace, userspace1, and tempspace1.
- The space for a TS in UDB is allocated in terms of the number of containers.
- A container is a physical storage place that could be a raw device DASD drive, an actual file, or a directory.
- While in OS/390 one can think of the tablespace as a physical object that maps to a linear dataset with a strict naming convention, in UDB the tablespace is a logical space object. It does not map to anything physical. The physical disk space allocated for this logical tablespace object in UDB is called a container.
- Containers store data and indexes in disk pages of 4KB, 8KB, 16KB, or 32KB sizes.
- The easiest way for an OS/390 DBA to understand the container is to think of it as being just like a dataset in MVS. In MVS we have a TS and it maps to a dataset, which happens to be linear VSAM; so in UDB we have a TS. This TS needs disk space, so we call that a container just as in the case of an MVS dataset. In MVS the dataset can be of different kinds, such as PDS, sequential dataset, linear VSAM dataset, etc. It's the same thing with containers; we have a directory container, file container, or raw device container. And as MVS datasets have extents, so in the container we have extents too.

- The physical files containing table data and index data have the following extensions respectively, (.DAT) and (.INX), and both can reside in the same physical container of the same logical TS.
- There is no indexspace concept in UDB as with OS/390.
- The TS itself can be two kinds: SMS managed (ie system managed), or DMS managed (ie database managed).
- The allocated containers for the default SMS-created TS are AIX directories that contain table data files and index files.
- One has the option of choosing the path where the TS files reside or just leaving it to the system to choose the default path and the directory for the files.
- Here are some examples of creating a database and tablespaces showing their containers:

```
CREATE DATABASE Nick
  CATALOG TABLESPACE
    MANAGED BY DATABASE USING
       (FILE 'C:\CAT\CATALOG.DAT' 2000
        ,FILE 'D:\CAT\CATALOG.DAT' 2000)
  TEMPORARY TABLESPACE
    MANAGED BY SYSTEM USING
       ('C:\TEMPTS'
        ,'D:\TEMPTS')
  USER TABLESPACE
    MANAGED BY DATABASE USING
       (FILE 'C:\TS\USERTS.DAT' 121)
```

- One can also issue a separate TS DLL independent of the CREATE DATABASE, but one has to connect to the database first. Here is an example of that:

```
$DB2 CONNECT TO Nick
$DB2 CREATE TABLESPACE DMSØ1
   MANAGED BY DATABASE USING
     (FILE 'C:\DMS\DMSØ1' 29)
```

Here is another example of that:

```
$DB2 CONNTECT TO Nick
$DB2 CREATE TABLESPACE DMSØ1
```

```
MANAGED BY DATABASE USING
  (DEVICE 'E:' 8001)
```

- It was mentioned that in OS/390 SMS can manage the dataset placement in the TS. In UDB, however, SMS does more than that; it is more sophisticated. In addition to allocating data files SMS handles space within these files. SMS does this dynamically by allocating or shrinking space for growth, as required.
- Once connected to the database, one can find out all the TSs in this database, their type and their related information by issuing:

```
$DB2 LIST TABLESPACES SHOW DETAIL
```

- One can also find out what types of container and how many of them are in each TS in this database by issuing:

```
$DB2 LIST TABLESPACE CONTAINERS FOR x SHOW DETAIL ;
```

  where *x* is the TS id picked up from the previous command, which means that one has to issue the list tablespaces command and pick up the specific TS one is interested in, then run the list containers command.

- One can also interrogate the catalog of this database and get some information. Here is a sample of how to do this:

```
$DB2 SELECT TBSPACE, TBSPACETYPE, TBSPACEID, DATATYPE,
     EXTENTSIZE, PREFETCHSIZE FROM SYSCAT.TABLESPACES
```

- In addition to the SMS TSs that are created and managed by the system there is another kind of TS, which is called the DMS TS.

  The DMS TS can use files or raw devices as containers. In this type of TS it's the DBA's responsibility to specify the file, its size, and the raw device. This type of TS more resembles the Pri and Sec quantities one is accustomed to when creating TSs in OS/390.

- In UDB up to Version 7, only type I indexes were supported. Now in UDB Version 8 type II indexes are supported, just as with OS/390.

- There are two types of table in UDB:
  - system tables, which are in the catalog of every database and have a qualifier of SYSIBM, SYSCAT, or SYSSTATS
  - User tables: permanent tables that hold permanent data, and temporary tables, called global temporary tables, that hold temporary data.
- The concepts of triggers, row id, identity column, MQT, and sequences apply also to UDB, as with OS/390.
- In UDB one cannot join two tables via SQL in one statement if the two tables are in two different databases because the two tables potentially can have the same name – because they are in totally independent databases.
- This is unlike OS/390, where one can join any number of tables in various databases because the full name of the tables must be unique within the entire DB2 subsystem.
- It was stated that in UDB one cannot join two tables in two different databases in one SQL statement. That is true unless, of course, the two databases are federated. Federation can involve another UDB DB2, SQL Server, or Oracle, etc. Federated databases are outside the scope of this article.

FILESYSTEMS

OS/390:

- The concept of filesystem is not used directly by DB2 OS/390. However, Unix System Services (USS) in OS/390 and z/OS utilize the filesystem concept. The OS/390 DBA will encounter the filesystem concept sooner or later, particularly with ERP applications such as PeopleSoft or SAP. Of course, UDB on AIX uses the filesystem concept to allocate its space. Therefore it is important for the DBA to understand the filesystem concept.

- Filesystem is a Unix concept of an inverted hierarchical tree structure that has an 'infinite' number of branches (called directories). Each branch can sprout any number of leaves (called files). In a strict Unix sense, directories are files but do not hold data. The directories' job is to maintain pointers to where files are located. Directories do this maintenance of pointers to the real data files through what are called inodes. Each directory has its own inode table. Even the Unix operating system itself has something called a superblock file where it maintains all the inodes of all the directories in the entire system.
- One can think of each directory in the Unix system as a mini filesystem managing files underneath it. One can also think of the whole Unix system as consisting of several filesystems integrated together to appear as a single filesystem, called the root filesystem.
- The file concept in Unix is supreme. Everything in Unix is thought of as a file. Even physical devices such as DASD, diskettes, CD-ROM, and memory are considered files.
- There is a concept in filesystems called mounting and unmounting of a file system. Mounting is an instruction to Unix to associate the filesystem (ie the hierarchical management of files as explained above) with the device name mentioned in the mount command instruction itself. This device name is a directory created by the Unix **mkdir** command. This device name directory is known as mountpoint and it is in here that the data resides and is managed by the filesystem. Dismounting is the reverse of mounting.

  It is difficult for the relational OS/390 DBA to understand the filesystem concept but it will become clearer with usage.
- Unix System Services, which is Unix on OS/390, has all the rules and structures of the Unix system. In order to accommodate the hierarchical filesystem of Unix with the MVS concept of dataset, IBM came up with a new type of dataset, called the Hierarchical File System dataset (HFS).

- The HFS is an MVS dataset that is used to store a filesystem (not the data). It is allocated to MVS as TYPE=HFS and is managed by DFSMS like any other MVS dataset. It is a single volume dataset but can reside on the same volume as other MVS datasets. It cannot be opened by TSO; it can be opened only by OpenEdition MVS kernel operation. It can have up to 123 extents. The size of the file in the filesystem is limited to 2GB. It can be referred to by JCL, which is a relief for us MVS 'old-timers'.
- The mount points in OS/390 USS are defined in SYS1.PARMLIB(BPXPRM*xx*).
- Here is an example of a filesystem mounting command in OS/390:

  ```
  $mount SYS1.UDBØ3Ø.HFS.USER  /u/udbØ3Ø
  ```

  This says associate the file system contained in *SYS1.UDB030.HFS.USER* HFS dataset with the data residing in this device */u/udb030*, which is a directory that was created by the Unix **mkdir** command.

UDB:

- UDB takes advantage of the AIX filesystem concept (as explained above) for space allocation and file management. (Remember DB2 on OS/390 does not use the filesystem concept. See note above.)
- Here is an example of a mounting command in Unix:

  ```
  $mount /dev/cdrom   /cdrom
  ```

  This says associate the file system contained in */dev/cdrom* with the data residing in device */cdrom*, which is a directory that was created by the Unix **mkdir** command.
- One can see that all the mountpoints defined in Unix are defined in *etc**mount*\. It can can be listed.
- One can see all the filesystems defined for the entire Unix in *etc**filesystem*\. It can be listed.

LOGGING MECHANISM

OS/390:

- Logging is important to the integrity of the data. It is necessary for roll-back (in case a transaction abends before it commits) and for roll-forward recovery to apply log records after a restore of a full image copy.
- There are active logs and dual active logs.
- There are archive logs and dual archive logs.
- The active logs, their duals, archive logs, and their duals belong to the entire DB2 subsystem, not to one database, as is the case in UDB.
- DB2 OS/390 uses a BSDS dataset to keep track of its logs, whether active or archived log datasets.
- DB2 OS/390 defines the characteristics of its log datasets in the DSNZPARM module.
- The OS/390 DBA needs to understand the logging concept and how it impacts running the various DB2 utilities such as Load, Reorg, Recover, etc.

UDB:

- Logging in UDB assumes the same functionality and importance as in OS/390.
- In UDB there are active logs, dual active logs, archive logs, and dual archive logs. All these logs are per database and not per instance (DB2 subsystem), as is the case with OS/390.
- The characteristics of the active and archive logs are stated in the db cfg file (compared with the DSNZPARM in OS/390). All these log files are physically located in a default directory called sqlogdir within that particular database.
- Dual logging is enabled by a parameter called mirrorlogpath in db cfg.

- One can change the default log path to a new name as follows:

```
$db2 "update db cfg for <database name> using newlogpath <new log path>"
```

- The equivalent to the BSDS dataset of OS/390 in the UDB environment is a file called sqlogctl.lfh. This file can be found under the UDB database directory sql00001.
- The number of these log files and their sizes are controlled by the parameters logprimary, logsecond, and logfilsiz in the db cfg file.
- The UDB DBA needs to understand the UDB logging concept and its impact on running utilities such as Backup and Restore etc. Here is a high-level elaboration of UDB logging.

  There are two kinds of logging in UDB:

  – Circular logging:

    - A number of active logs are created automatically when a new database is created (the number and size are specifed in db cfg). These active logs are used in a round-robin fashion by UDB. When one log file is filled, the next log file is used, and the process continues until all the logs are filled. When all the active logs are filled, the process will wrap around to write over the first log file. If the first log file still has some transactions that did not commit, then UDB will dynamically create a secondary log file to hold the log records until, hopefully, all the transactions in the first file are committed, else a second secondary file is created, then a third secondary file is created. All the logs, primary or secondary, are considered part of the active log. It is worth repeating that the number of the primary and secondary files and their sizes are defined in the db cfg file. The dynamically created secondary files will be removed automatically when they are not needed, ie after transactions commit. Sometimes people refer to circular logging as

LOGRETAIN=NO logging, LOGRETAIN being a parameter in the db cfg file.

- Circular logging is the default when a new database is created and is most suitable for crash recovery.
- Circular logging is not suitable for recovering long-running transactions because it does not allow roll-forward recovery, ie restoring a full image copy and applying the log records. For that we need archival logging.

– Archival logging:

- There is nothing special about archival logs; they contain the same log data except that the logs are not reused, and, when an active log is filled, a new one is created in the database log directory. This process goes on, so when we need to recover a database to current, we just restore the appropriate full image copy and apply the logs.
- Archival logging in UDB in principle is similar to OS/390 except in OS/390 the archiving process occurs automatically. In UDB, however, the archiving process does not kick in unless a user-exit is enabled to copy the filled active log to tape or to another location on DASD. If one decides to archive to the same log directory on DASD, that is called an on-line archive. If one decides to archive the active log somewhere else, that is called an off-line archive.

SYSCOPY AND SYSLGRNX VERSUS HISTORY FILE

OS/390:

- SYSIBM.SYSCOPY is an OS/390 catalog table used to keep track of image copies and other various activities needed for recovery.

- SYSIBM.SYSCOPY coordinates its activities with another catalog table, called SYSIBM.SYSLGRNGX.
- One can interrogate the SYSIBM.SYSCOPY via normal SQL.
- The SYSIBM.SYSCOPY and SYSIBM.SYSLGRNGX tables can grow in space a lot after a period of activity.
- The DBA needs to use the MODIFY utility with the RECOVRY option to delete aged entries from both tables.

UDB:

- The equivalent to SYSIBM.SYSCOPY and SYSIBM.SYSLGRNGX tables of OS/390 in UDB is the HISTORY file (*DB2RHIST*), which is listed under the database directory SQL00001.
- One can interrogate this file by the **LIST HISTORY** command.
- As in the case of OS/390 this HISTORY file grows after a period of activity so one can use the **PRUNE HISTORY** command to delete aged entries.

UTILITIES

Only the following DB2 utilities will be contrasted: image copy utility, recover utility, unload utility, and reorg utility.

OS/390 (image copy utility):

- It is known in OS/390 as the image copy utility. It can take copies of TSs and copies of indexes.
- It does not take copies of the entire database as in UDB.
- It can take a full image copy or an incremental image copy. If incremental image copy is taken, one needs to run a mergecopy utility to merge the last full image copy with the latest incremental image copy.
- One needs to specify the dataset name in the JCL where the back-up of the TS or Idx is to reside.

- It can take image copies on-line and off-line.
- Image copies can be utilized by the recover utility, the unload utility, the load utility, and the mergcopy utility.
- If full image copies of indexes are desired, the index in question should have been created with the image copy option enabled. Be aware that incremental image copies are not permitted on indexes. Only full image copies are allowed for indexes.

UDB (image copy utility):

- In UDB the image copy utility is known as the **backup** command.
- It copies the entire database or selected TSs. It is not restricted to copy only the TSs as is the case in OS/390.
- It can take a full image copy or an incremental image copy.
- In UDB one specifies the directory where one wants the back-up of the entire database or of the TS to reside.
- It can take image copies on-line and off-line. Image copies are used by the restore utility, export utility, import utility, or load utility.

OS/390 (recover utility):

- The OS/390 recover utility recovers TSs, not databases as in UDB. It does not recover a single table unless the table is in one single TS.
- The recover utility recovers image copies that are registered in SYSIBM.SYSCOPY. The equivalent in UDB is the history file. DBAs can interrogate both files for recovery purposes. The recover utility coordinates the recovery process for the needed logs (whether active logs or archived ones) with the help of SYSIBM.SYSLGRNX and the BSDS dataset.
- The OS/390 recover utility also recovers indexspace provided the image copy option of the index at index

definition time was enabled. Since incremental image copies are not permitted for indexes then full image copies are needed for the recover utility to recover indexes.

- In addition to recovering TSs, the recover utility can recover single DB2 partitions, single DB2 pages, or ranges of DB2 pages-in-error.
- The OS/390 recover utility can recover the DB2 catalog and directory, but the steps within the recovery JCL job should be ordered in a strict sequence as explained in the *Administration Guide*.

UDB (recover utility):

- The equivalent to OS/390 recover utility is the **RESTORE** command. It restores the entire database or selected TSs from full image copies then it applies the logs.
- Actually the **RESTORE** command has the logapply phase in it.
- However, one can issue the **RESTORE** command and withhold applying the logs to the restored image copy. Why would we do that? The DBA does that because he/she wants to issue the **ROLLFORWARD** command to a specific timestamp. This timestamp acts like an RBA in an OS/390 environment.
- The **ROLLFORWARD** command will apply the log records to the restored full image copy, but in a deferred time of one's choice.
- Image copies of UDB are registered in the history file. The logs are recorded in the SQLOGCTL.LFH file, which is equivalent to the BSDS in OS/390. One can interrogate the history file and see the image copies that are registered in it and see their timestamps, etc.
- UDB **RESTORE**s can be performed for databases in off-line mode only, while it can be performed in off-line and on-line mode for tablespaces.

- **ROLLFORWARDs**, however, for databases can be performed only in off-line mode.
- UDB **RESTORE** for the catalog is not a big deal like it is in OS/390 because the catalog in UDB is just like any other tablespace.
- Be aware of the two types of logging in UDB and its interaction with the **RESTORE** command.

OS/390 (unload utility):

- The DSNTIAUL stand-alone unload utility has no equivalent in UDB.
- The official OS/390 UNLOAD utility unloads data from individual tables, from entire TSs, and from image copies. When the unload utility unloads several tables from a TS in one unload dataset, it prefixes every unloaded record with a header identifying it with its mother table.
- The unload utility can unload selected fields from the table record.
- The counterpart of the unload utility in UDB is the **EXPORT** command.
- The UNLOAD utility also creates the load parameters for the table or tables should one want to load back the DB2 tables.
- One can use LISTDEF and TEMPLATE features to save time.
- Remember that the OS/390 UNLOAD cannot unload the entire database, but in UDB, as you will see below, one can use the **DB2MOVE** command with the EXPORT option to unload the entire UDB database.

UDB (unload utility):

- The equivalent of the OS/390 UNLOAD utility in UDB is the **EXPORT** command and the **DB2MOVE** command.
- The **EXPORT** command uses SELECT statements, thus it can unload only one table at a time. Before one invokes the

EXPORT command one has to CONNECT to the specific database in question. Here is an example:

```
$DB2 "CONNECT TO nick"
      $DB2 "EXPORT TO nick_output.del OF DEL MESSAGES msgs.txt
        SELECT * FROM udb030.products(columnname1,columnname2,etc)"
```

- The **DB2MOVE** command with EXPORT option can take away the limitation of unloading one table per **EXPORT** command. The **DB2MOVE** can unload several tables at one time. Actually one can unload the entire database using the **DB2MOVE** command with the EXPORT option. Let us elaborate on this **DB2MOVE** command:

```
$DB2 "CONNECT TO nickdatabase"
      $DB2 "DB2MOVE nickdatabase EXPORT"
```

  This produces the following files by default:

  – an *export.out* file containing the result of the export action

  – a *db2move.lst* file containing the list of files and the source table names

  – tabnnn.ixf are a series of files containing data from the tables

  – tabnnn.msg are a series of files containing the messages associated with the exported table.

- The dynamic LISTDEF and TEMPLATE features of OS/390 unfortunately do not have equivalents in UDB.
- The unload utility in UDB supports the DEL, IXF, and WSF file types.

OS/390 (reorg utility)

- The purpose of the reorg utility is to optimize the physical distribution of data in the TSs and indexes. This is needed because the volatility of the data may have caused disorganization and fragmentation within the data. When there is a lot of fragmentation there are lots of I/Os, thus performance deteriorates.

- In OS/390 one reorgs a TS and not a table, as is the case in UDB.
- When one reorgs a TS in OS/390, the indexes on all the tables in that TS are organized too.
- One can also reorg a TS or an index separately if desired.
- One can also run the reorg on-line or off-line.
- Make it a habit to run the runstats utility after every reorg to pick up the latest statistics.
- In OS/390 there is no utility to inform the DBA when to reorg a particular TS. In UDB there is a utility called REORGCHK. However, some people can argue that in OS/390 the REPORTONLY option of the reorg utility is the equivalent of the REORGCHK in UDB!

UDB (reorg utility):

- The purpose of the reorg command in UDB is the same as in OS/390, which is to optimize the physical distribution of data.
- In UDB one reorgs a table and not a TS, as is the case in OS/390. When reorging a table in UDB it is important to use the fully-qualified name of the table.
- When one reorgs a table in UDB all the indexes built on that table are organized too.
- There is no on-line reorg index command in UDB as with OS/390.
- Indexes in UDB can be automatically reorganized all the time, if one wishes, by specifying the parameter MINPCTUSED with a value larger than zero when defining the index.
- When one reorgs a table in UDB that has multiple indexes, the table will be organized according to one of the indexes that was defined as cluster. However, in UDB one can specify that the table be reorged according to a particular index. This may be needed if an application uses this particular index a lot.

- Here is an example of a UDB reorg command:

```
$DB2 REORG TABLE udb030.tablename
```

- With the coming of index type 2 in UDB Version 8, a UDB table can now be reorged on-line just as with OS/390. In previous UDB versions only an off-line reorg for a UDB table was possible.
- It is useful to run the **REORGCHK** command before running the **REORG** command on a UDB table. The **REORGCHK** utility calculates statistics on the database to determine whether tables need to be reorganized or not, thus saving the time of an unnecessary reorg. Interestingly, the **REORGCHK** utility invokes the **runstats** utility under the cover to obtain the latest statistics so that the **REORGCHK** utility can do its calculations and produce accurate recommendations.
- Make it a habit to run the **runstats** utility after every reorg to pick up the latest statistics.

Editor's note: this article will be continued in future issues.

Nicola Nur
Senior DBA (Canada)

Virtual Frame Buffers

Editor's note: this article takes a light-hearted look at a difficult subject. We hope our regular readers will not mind this one-off departure from our usual style, and will find the information presented to be of interest.

"Virtual what . . . ?", I hear 99% of you say. Rather than endure a dry and less than exciting explanation at this stage, which will be totally meaningless, consider the following scenario.

Act 1, Scene 1: The deepest and darkest recesses of a machine room in the Pots of Money, or POM, company Ltd,

where a contractor (or consultant, or indeed any other member of the Guild of Daylight and Highway Robbers, or its sister organization, Irrational Brigands and Mercenaries – take your pick) is approaching the end of a thankless project and is reading the latest edition of *Playboy* while trying to hide himself from the clutches of a number of demanding POM staff who have the AIX skills and puzzle-solving capabilities of a three-year old dyslexic troll.

In walks Downright Stupid, stage right, a grossly overpaid application architect with the brain of a retarded gerbil. “Ah, Tonto, there you are. Found you at last. Just checking out the machines, eh?”, enquired Downright, who had been searching for Tonto for the last three hours.

Had Downright realized that consultants rapidly acquire the capacity to withstand cold as easily as a polar bear, and that machine rooms are their favourite hiding places, particularly ones which resemble the dusty and overcrowded recesses of the basement of the British Museum with nooks and crannies known to only the most persistent of intrepid adventurers, he could have saved himself at least 2 hours and 59 minutes of fruitless searching.

Tonto, using his oft-practised legerdemain, deftly hid the *Playboy* inside a seemingly respectable AIX manual and sheepishly replied in his usual articulate fashion, “Er, yeh. Wodja want?”.

“Well, Tonto,” continued Downright in his finest Cockney accent, “we ’ave a problem with an application that uses X-Windows. It used to work, and we ’aven’t made any changes, ’onestly, but after our now departed system administrator, the ’onourable ’arbinger ’opeless from ’ounslow, upgraded the system, we can’t get it to work. It’s cream crackered, guv.”

“What alliterative madness is this?”, thought Tonto, and his sharp intake of breath would have done a pair of giant reverse bellows proud as his heartbeat leapt into overdrive when he realized that he knew virtually nothing about the well-named unknown X. Like all system administrators, when first learning

AIX he read all the manuals, absorbed about 1% relating to X11 ("Is it still called that? Must be at least 12 or 13 by now", he pondered), played around with **xclock**s, childishly displaying them on the terminal of anyone who foolishly logged on at the same time, modified the CDE desktop before realizing that psychedelic puce, which, although it brought back fond memories of youthful experimentation with alternative substances, was a most unattractive colour for most sane and sober users, resulting in hallucinations followed by manic depression after staring at a screen for anything more than 30 seconds, and then promptly forgot nearly all of it since it had almost no relevance to anyone other than the most determined and demented X-Windows programmer.

"Oh yes. And what exactly is the problem, Downright?" he nonchalantly replied, hoping to elicit a little more information than that supplied to date, and at the same time successfully hiding his distinct unease while surreptitiously mopping a brow which was rapidly acquiring large droplets of sweat despite the apparent sub-zero temperature.

Downright, not renowned for his powers of observation, with childlike naivety ploughed on. "When I log on to our server from my machine, I execute our credit application using the following commands:

```
export DISPLAY=stupid:0
cr_app
```

But I keep getting the following error messages:

```
Unable to open display stupid:0
```

So what do I do?"

Resisting the temptation to tell Downright that he should give up computing completely and commit ritual Hari Kiri, or, equally distasteful, become a manager, Tonto realized that the only knowledge of X-Windows that he, Tonto, still possessed might seriously handicap his attempts to convince Downright and his POM colleagues that he was indeed the guru he professed to be. He was a firm believer in the contractor/consultant's universal

eternal maxim number 1: *Provided you can convince the customer that you know more than he does without actually telling him what you know, then you have him at a disadvantage*, and so he replied, “Give me five minutes”.

Away from the prying eyes of Downright (using universal eternal maxim number 2: *Under no circumstances show the customer what you are doing since he may discover it is so simple that he didn’t need you in the first place*), Tonto swiftly logged on to the server from the console as **root** and entered the command:

```
xhost +
```

“See if that helps”, said Tonto.

“Works a treat”, said Downright when he returned after a couple of minutes, “What did you do?”

Tonto, resisting the urge to tell the Downright that he, Downright, was an inferior being and that he, Tonto, with his infinite wisdom and godlike guru status had solved the problem because of superior brainpower and experience, instead said, “The discombobulator had been disengaged due to fluctuations in the earth’s magnetic field caused by an excess of virtually nothing. I merely remedied this by running the engage combobulator program. Simple when you know how.”

“Knew it was nothing to do with my application’, hazarded a severely confused Downright. “Lucky you were ’ere when this camelbob thing failed then”, continued the grey matter-deficient architect.

Act 2 Scene 1: Two hours later Tonto is still in the machine room, which is a mere 2 degrees above absolute zero, frantically turning over the dog-eared pages of his *Playboy* magazine to keep warm and prevent his body from acquiring super-conductivity status, the console screen being pitched at the perfect angle to warn him of the stealthy approach of any POM punter foolish enough to require his assistance.

In walks Thick As Two Short Planks stage left, or ThATS Planks

as he was commonly known, a sales colleague of Downright Stupid marginally more intelligent than a decomposing hippopotamus (although surprisingly possessing the cunning and mercenary skills required of his particular profession), almost catching Tonto in an embarrassing position.

“Is that you, Tonto?” says ThATS, having worked with him for the last six months and obviously having problems with recognition, perhaps because of hallucinations brought on by a psychedelic puce desktop. “I’ve got problems with the same application as Downright. When I log on to the server and execute **cr_app**, I get the same error messages that he had.”

Tonto, clutching at the proverbial straw and rapidly trying to think of anything to put off performing an in-depth investigation, his mind still on a particularly fascinating and thought provoking article he had been reading about the mating habits of drunken Glaswegians, phones Downright Stupid and asks, “Seems we may have further unforeseen problems with the discombobulator; can you still run your application OK?”

After receiving something approaching a Neanderthal grunt, which Tonto assumed was the Post-Cambrian equivalent of a ‘yes’, he quietly panics and immediately thinks, “Oh sugar, what have these zombies been doing?” Having graduated with first class honours in Trickery, Treachery, Skulduggery, and Procrastination in the University of Life, he quickly informs ThATS, “Sounds like the ancillary twangle of the discombobulator vortex failed. I’ll check it out and get back to you.”

Away staggers ThATS, stage right, knuckles scraping the ground, his tiny mind focused on the deceit he is about to perpetrate on the next unsuspecting customer.

Some time later Tonto reluctantly realizes that, in addition to the customer staff indeed being a bunch of morons, he unfortunately had sunk to their levels and made a serious blunder by running **xhost +** and then logging off from the console. This did not prevent Downright from continuing to run the application from his machine, but prevented other users who later logged on to the server, such as ThATS, from running **cr_app**.

To confirm his suspicions, he logged on again as **root** and ran **xhost**, only to discover the output:

```
Access control enabled: only authorized users allowed access
```

In the meantime, to keep the punters happy, he logs back in as **root**, runs **xhost+**, and leaves his session logged in. “That should keep them quiet for a while”, mutters Tonto to no-one in particular, but mainly into his beard, wiping away disturbing bits of froth appearing at the corners of his mouth.

“So, does this mean”, he mused, “that **cr_app** requires **root** to have a permanent window open on the console, with access control permanently disabled, and not be able to log off, otherwise access control is re-enabled? Some research required here”, he thought miserably, listening to the increasing rumbles from his nether regions and realizing that he would have to forgo his normal two-hour gourmet buffet lunch, which was, of course, chargeable to the customer at the usual time and a half.

Act 2, Scene 2: Tonto, still in the machine room and looking in extreme discomfort because of missing his eagerly anticipated, but now rapidly vanishing, lunch, is seen poring over a document which he has discovered completely by accident after spending two hours trawling through various cunningly-concealed IBM Web sites, none of which seemed to provide anything he could remotely understand; were it not for a one-in-a-million chance, he would still be searching for the illusive information.

In fact he had become so disheartened by the apparently futile search that his mind had started to wander from the task in hand, increasingly gravitating towards food and his missed lunch. After two hours he was so desperate for victuals that he decided to see whether there were any local companies who could deliver a buffet lunch, having by this time given up hope of leaving the office to satisfy his gastronomic cravings, but when he typed ‘buffet’ in the Google search panel, his frozen digits inadvertently entered ‘buffer’.

The consequence of this not so freakish freak of nature was that he was eventually presented with a document with the

catchy title *Appendix E: The X Virtual Frame Buffer*, the Appendix apparently being an attachment to the *AIX-Windows Programming Guide*.

Aarghh! The mere mention of Windows programming was enough to bring on palpitations and set Tonto's knees atrembling. Now most people presented with such a document would immediately click on the 'Back' button and breathe a sigh of relief at having so easily escaped a fate worse than death, but Tonto was made of sterner stuff, and tiny bells had begun ringing in the deepest recesses of his mind as under-utilized neurons started to make the tenuous, but still unknown, connection between his previous use of **xhost**, and the **X** of the Appendix title.

As he read the first few paragraphs, however, he realized to his horror that the document was phrased in exquisite gobbledygook, and typical IBM technicaleze, replete with acronyms, synonyms, antonyms, and meaningless abbreviations, where 10 long, complicated, and often made-up, words are used instead of a single simple one. "The Plain English Society", an organization renowned for its intolerance of gibberish and officialeze, "would award a Gold Medal with knobs on for this one", thought Tonto.

He re-read the document 15 times, trying various juxtapositions of verbs, nouns, and adjectives, and finally coming to the conclusion that, after stripping out all the meaningless and repeated phrases, junk and marketing hyperbole, there probably was some good stuff in there, and it might actually be the solution to his seemingly intractable problem.

After many 'What?'s, 'Why?'s, 'How?'s, and 'You must be joking's, he had discovered that the XVFB, as it is lovingly referred to, creates a private 3D rendering area on a server that is used by an application accessing the server remotely, so that X-Windows images can be displayed on the remote machine rather than locally on the server. The method by which XVFB is started, coupled with the way in which the remote application is designed and run, appeared to be independent of any local login

session, and so gets round the automatic re-enabling of access control when the server login session which disabled it (using **xhost+**) finishes. “Gripping reading”, thought Tonto.

He further discovered that XVFB was particularly useful in environments where a graphics adapter was not present on the server. With no physical graphics device it is not possible to generate RGB signals and therefore impossible to connect a monitor to a system to view the contents of the Virtual Frame Buffer. The frame buffer in such a configuration is stored in system memory and all graphics processing is done in the software using the CPU, thus allowing rendered images to be distributed across a network and viewed remotely on systems that have a graphics capability. By this stage Tonto was marvelling at his rapidly acquired précis skills.

“Better do some experimenting and testing”, muttered Tonto, getting quite excited at the prospect of solving his problem and temporarily forgetting about his missed lunch; after all, he really was a bit of a nerd.

“So, let’s first load the XVFB into the X server and start the X server without using any installed graphics adapter.” He read from the document without moving his lips:

```
/usr/bin/X11/X -force -vfb -x abx -x dbe -x GLX :1 &
```

“Wonder what all those options are?” questioned Tonto, but apart from the **–vfb** flag, which actually loads XVFB into the X server, he neither knows, nor cares, assuming that if it says so in the document, then it must be correct. He had, of course, forgotten universal eternal maxim number 3: *Any statement in an official IBM technical document has more than a 50% chance of being inaccurate*.

He could have used any display number apart from **:0** (since this display number is used by the installed graphics adapter), but he chose **:1**. ‘Hmm, seems that I can run multiple instances of the XVFB with different display numbers,” he astutely notes, but then wonders why anyone would choose to do so.

“Better test it out and see if it’s working and IBM hasn’t discombobulated it with the latest maintenance level”, he ponders wisely, but confusing even himself. Years of frustrating experience had taught him that it required only some apparently meaningless, and only partially-tested, change to code by a programmer in a totally unconnected but recently upgraded fileset to have the direst consequences to applications previously running without a problem.

After checking that the XVFB was indeed still running and hadn’t suddenly decided to call it a day and kill itself off in the time-honoured fashion of a less than robust competitive operating system, Tonto ran **xclock** on his new display:

```
xclock -display :1 &
```

Nothing, of course, appeared on his console screen since that was still displaying **:0**.

On another machine running an X server and connected to the network, he now ran **xhost +** to allow clients to connect to its X server. Visibly weakening from a lack of food, he staggered back to the XVFB machine and now ran a command with which he was totally unfamiliar, **xclock**, having just about exhausted his knowledge of X commands. “According to the documentation, if I run this command I should get the window ID for the **xclock** client. Here goes.”:

```
xwininfo -root -tree | grep xclock
```

Nothing, zilch, not a dicky bird. A quick look at the command line options used by **xwininfo -?** revealed to Tonto the truth of universal eternal maxim number 3. The command should have been:

```
xwininfo -display :1 -root -tree | grep xclock
```

Which now produced the window ID **0x400009**. “Now I have to display the client window of the XFVB system on the other machine.’ And Tonto types in a command line which, according to the documentation, should now display the **xclock** on the remote machine:

```
xwd -id 0x400009 | xwud -display other_system:0.0
```

A quick check reveals no **xclock** and after surprisingly finding **man** entries for both **xwd** and **xwud**, and realizing that maxim number 3 has kicked in again, he retypes the command:

```
xwd -display :1 -id 0x400009 | xwud -display other_system:0.0
```

At long last, success; the **xclock** appears on the remote system. His normally inquisitive nature, and his talent for extracting sense from apparent nonsense, reveals that **xwd** dumps the image of an enhanced X-Windows window, and the **xwud** command then displays the contents of the dump output for the specified window ID, ie that of the **xclock**, on the remote system.

“OK, theoretically this XVFB thingummy should solve the problem”, murmurs a sceptical Tonto, remembering universal eternal maxim number 4: *What works in a test environment often goes horribly pear-shaped when the customer gets his hands on it*.

“All I have to do now is make sure the XVFB starts when the system boots up”, and so he kills off the X server with the **–vfb** flag he started earlier and then makes an entry in **/etc/inittab**:

```
xvfb:2:respawn:/usr/bin/X11/X -force -vfb -x abx -x dbe -x GLX
                                                        :1 >/dev/null
```

After running **telinit q**, the XVFB is started and, he logs off and logs back in to reveal that it is indeed still running.

“Now for the mor mor mor morons”, he stutters because of increasing cold, tightening his metaphorical belt in an attempt to control his increasingly spasmodic breathing brought on by his lowered blood sugar levels and a rising nervousness that what he had just tested might fail miserably because of maxim number 4. “Just hope that this application of theirs can cope with **:1** and the virtual frame buffer. Wonder what the ’onourable ’arbinger did during his upgrade?”, he speculates, but not particularly caring one way or the other.

After locating the disastrously disturbed duo, Tonto asks both

of them to log off and try to run **cr_app** again. Squeals of delight tell Tonto that, for the time being, the XVFB is the bees' knees. "What did you do?" grunts Thick As Two Short.

"Shall I tell him the truth, or shall I perpetuate the myth of my genius?", muses Tonto. But then he remembers universal eternal maxim number 5: *Never explain complicated technical procedures to a customer since there is a 95% certainty that he will not understand; or, if there is the remotest possibility that he will understand, fudge the details otherwise he will acquire The Knowledge and you might be out of a lucrative contract and thus unable to retire at the age of 35.*

"I had to change the display to one of affection otherwise the discombobulator enters the bobulator phase leading to the uncontrollable spreading of sinus waves", lies Tonto skilfully and quickly, wishing to perpetuate the myth of his invincibility and realizing that it would take him too long to tell the truth, not that he had the inclination to impart his hard-earned knowledge with someone wallowing in the shallows of a seriously depleted gene pool.

ThATS, having a bad time with this total gibberish and wondering why Tonto has broken into Swahili, finally gives up the struggle and says, "Whatever would we do without you?".

"Indeed, yes, indeed", agrees Tonto.

Finale: Several days later in the office of Peerless Backstabber, the exceedingly brash, opportunistic, ambitious, sycophantic, but decidedly non-technical, manager of the IT department, an organizational genius at meaningless, pointless, endless, and superfluous meetings, but apart from that, totally useless. Opposite sits a smug Tonto, sure in the belief that his lucrative contract is about to be extended and that this particular tête-a-tête is still racking up the pennies.

"Well, my boys [*as Peerless affectionately refers to the POM staff who make him look like the next best thing since sliced bread*] think the sun shines out of your adverbial", states Peerless in his usual jocular but malapropos fashion. "Good job

you are around otherwise we would have no end of problems."

"Must be my years of experience and an inexhaustible supply of inquisitiveness, coupled with problem solving capabilities similar to those of that Hercule Parrot bloke", proffers Tonto, with only the merest hint of sarcasm, totally lost on Peerless.

"Just so", agrees the megalomaniac manager, and with a flourish signs a document on the desk in front of him and hands it to Tonto. "There you are then, since we can't do without you, that's another two-year contract."

Tonto picks up the road-to-riches and early-retirement extension, murmurs his thanks and after quietly closing the door behind him, leaps in the air, screams "Yessss!!!!", and then gleefully exits stage left, hopping from foot to foot singing in his finest tuneless baritone, "We're in the money, we're in the money…".

Richard Handforth
AIX and HACMP Software Specialist (UK)

Cleo Communications has announced Cleo TN3270 and Cleo SNA products for AIX 5L.

Cleo TN3270 is a Windows, Unix, or Linux solution for establishing 3270 connections to IBM mainframes or AS/400s. Cleo 3270 SNA is a combined software-hardware solution, which enables enterprise Unix servers to access legacy applications or data on IBM mainframes or AS/400s. Cleo 3270 SNA supports all features of the SNA protocol and supports multiple mainframe communications from a single server. The multiple connections allow several mainframes to be used in a single application as well as a convenient method for supporting disaster recovery and hot back-up configurations.

For further information contact:
Cleo Communications, PO Box 15835, Loves Park, IL 61132-5835, USA.
Tel: (815) 654 8110.
URL: http://www.cleo.com/news/press releases/AIX5L.asp.

* * *

MicroStrategy has launched an AIX version of 7i, its business intelligence software.

The new Universal Edition is compiled for 32-bit Windows and 64-bit Unix modes from the same code base so it can run on AIX, Sun Solaris, and Windows.

For further information contact:
MicroStrategy, 1861 International Drive, McLean, VA 22102, USA.
Tel: (703) 848 8600.
URL: http://www.microstrategy.com/ ?CID=1991IWplat.

* * *

Signiant has announced that it has expanded platform support of its Mobilize solution to include AIX. The product enables users to simplify the complex task of managing data across their remote IT infrastructures independently of server, storage vendor, or OS platform.

The Mobilize remote data management suite helps companies break from the cycle of backing up ever-growing amounts of data by identifying and profiling data on remote systems, automating policies to periodically archive data no longer being used, and intelligently consolidating data to a central location where it can be efficiently backed up to disk or tape.

For further information contact:
Signiant, 15 Third Ave, Burlington, MA 01803, USA.
Tel: (781) 221 0022.
URL: http://www.signiant.com/company_ news_pr_platform.htm.

* * *

Leapstone Systems has announced its CCE contentMANAGER now runs on AIX and WebSphere Application Server.

Leapstone's Communications Convergence Engine (CCE) products enable wireline, wireless, and cable operators to design, package, deploy, and manage a range of content and services.

The CCE contentMANAGER allows users to deliver applications and content over broadband networks and allows content from multiple sources to be included.

For further information contact:
Leapstone Systems, 220 Davidson Avenue, Somerset, NJ 08873, USA.
Tel: (732) 537 6800.
URL: http://www.leapstone.com/products/ content_manager.asp.