117

AIX update

July 2005

In this issue

A tc PUBLICATION

AIX Update

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

Colin Smith
E-mail: info@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.

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.

De-allocation of AIX system resources

De-allocation of system resources is helpful for benchmarking applications or trouble-shooting and reproducing application problems. It might happen that you have to de-allocate a CPU (reduce your system CPU) to simulate a performance problem with your application that your customers are reporting on their two-CPU system, which you cannot reproduce in your lab on your four-CPU system. Other uses for the de-allocation of system resources would be to get base-line benchmark numbers for your application's performance with various system configurations. It also allows you to dynamically take a failing CPU off-line.

In today's world of logical partitioning with LPAR and DLPAR, the same thing can be accomplished. But when you do not have the convenience of using logical partitioning DLPAR to de-allocate or allocate resources, this procedure will be handy. It will show you how to de-allocate CPU and memory from your system as if they had never been introduced into the AIX system, and how to re-allocate them.

We will start by de-allocating a CPU – but before we start, we need to know the hardware architecture of the platform you are running (whether it is PCI or micro-channel, MCA) because the procedures will differ. To determine your platform architecture, you can use one of the following commands:

```
bootinfo -p
```

or:

```
lscfg | grep "Model Architecture"
```

If the output from the commands is **chrp** or **rspc**, it means that your platform architecture is PCI. If the output is **rs6k** or **rs6ksmp**, the platform architecture is MCA.

For PCI systems, you use the following procedure to de-allocate CPUs; but remember that you need a system with a minimum of three CPUs to use the de-allocation procedure. Please note that the actual output from the commands may differ on your system from the examples:

- Check how many CPUs are physically on your system and how many are active:

  ```
  # lsdev -Cc processor
  procØ      Available ØØ-ØØ       Processor
  proc1      Available ØØ-Ø1       Processor
  proc2      Available ØØ-Ø2       Processor
  proc3      Available ØØ-Ø3       Processor

  # bindprocessor -q
  The available processors are:  Ø 1 2 3
  ```

- Note: to enable run-time processor de-allocation, you must change a kernel parameter called **cpuguard**. If the **cpuguard** kernel parameter is disabled, you will not be able to de-allocate CPUs successfully. The **cpuguard** requires a minimum of three CPUs:

  ```
  # lsattr -El sysØ -a cpuguard
  cpuguard disable CPU Guard True

  # chdev -l sysØ -a cpuguard=enable
  ```

- The command used to de-allocate CPU is **cpu_deallocate**, and it is part of the *bos.rte.control* fileset:

  ```
  # /usr/sbin/cpu_deallocate  cpu_id
  ```

  where *cpu_id* is the logical CPU ID. For more information on this command please refer to the command man pages.

  The logical CPU ID is not necessarily the same number that is associated with proc# in **lsdev -Cc processor** output. The proc#s are often not in sequence.

- After de-allocation of the CPU, you can check for available processors:

  ```
  # bindprocessor -q
  The available processors are:  Ø 2 3

              # lsattr -El proc2
  ```

```
state     disable          Processor state False
type      PowerPC_POWER3 Processor type  False
frequency 200000000      Processor Speed False
```

The deactivation is immediate. To reactivate the CPUs, the system must be rebooted.

For MCA systems, you can use the **cpu_state** command, which is more flexible and is part of the *bos.mp* fileset. You do not have to re-boot to reactivate the deactivated processor. Actually, if you reboot the system after deactivating a processor, this processor will stay disabled. Note that the following examples were taken from the command man pages because I do not have an MCA system to test on (because MCA platforms are very old and few are still in production) – but I included the procedure just in case.

```
cpu_state   -l | { -d |  -e } ProcessorNumber
```

where:

- **-d** – disables the specified processor.
- **-e** – enables the specified processor.
- **-l** – lists the status of all processors.

Examples:

- To list all processors:

```
# cpu_state -l
Name    Cpu  Status     Location
proc0   0    Enabled    00-0P-00-00
proc1   1    Enabled    00-0P-00-01
proc2   2    Enabled    00-0Q-00-00
proc3   3    Enabled    00-0Q-00-01
```

- To disable processor 1:

```
# cpu_state -d 1
# cpu_state -l
Name    Cpu  Status     Location
proc0   0    Enabled    00-0P-00-00
proc1   1    Disabled   00-0P-00-01
proc2   2    Enabled    00-0Q-00-00
proc3   3    Enabled    00-0Q-00-01
```

- To re-enable processor 1:

```
# cpu_state -e 1
# cpu_state -l
Name     Cpu  Status     Location
procØ    Ø    Enabled    ØØ-ØP-ØØ-ØØ
proc1    1    Enabled    ØØ-ØP-ØØ-Ø1
proc2    2    Enabled    ØØ-ØQ-ØØ-ØØ
proc3    3    Enabled    ØØ-ØQ-ØØ-Ø1
```

Since we mentioned the command **bindprocessor** before, it is important to know that this command can be used to bind or unbind the kernel threads of a process to a certain processor, in addition to the listing of available processors that we saw it doing earlier. You can use this command, which is part of the *bos.mp* fileset, to bind a process of your application to a certain CPU, therefore simulating a single CPU system. For example:

- To bind process with PID 339215 to CPU 0:

```
# /usr/sbin/bindprocessor 339215 Ø
```

- To unbind the same process from CPU 0:

```
# /usr/sbin/bindprocessor -u 339215
```

Not only can you deactivate a CPU, you can also deactivate memory. To change your memory configuration with AIX, you need to use the **rmss** command, which is part of the *bos.perf.tools* fileset. For more information on the **rmss** command, refer to the command man pages. You do not have to reboot your machine to go back to your original parameters:

- To print the current memory size, enter:

```
# /usr/bin/rmss -p
Simulated memory size is 2Ø48 Mb.
```

- To change your current memory size to 1024MB (meaning to deactivate 1GB of memory and use only half of the original memory), enter:

```
# /usr/bin/rmss -c 1Ø24
Simulated memory size changed to 1Ø24 Mb.
```

- To reset the memory size to the real memory size of the machine, enter:

```
# /usr/bin/rmss  -r
Simulated memory size is 2048 Mb.
```

Basim Chafik
Senior Systems Analyst
IBM Certified Advanced Technical Expert (CATE)
Plexus (Division of BancTec) (Canada)

Proactive AIX update tools and techniques

AIX operating system updates are a complicated and sometimes confusing activity. The reasons for this is the combination of a very high number of new system features that are introduced, requirements to support new hardware devices, and the need to fix bugs that are discovered in the system. This article will review the terminology used by IBM to classify the fixes and updates, and the tools that system administrators can use to integrate them on their computers.

TERMINOLOGY

IBM uses the following terms to describe the AIX updates.

PMR

A Problem Management Record (PMR) is a unique IBM ID tracking record for customer-reported problems.

APAR

An Authorized Program Analysis Report (APAR) associates a fix to a PMR. APAR numbers can be used to obtain the required fix. When documenting software requirements, it's best to list the APAR number rather than the PTF (described later) or PMR number. You will always be able to determine whether an APAR is installed on your system using the command **instfix -ivk APAR_NUMBER**, whereas installed PTFs are not traceable

PTF

A Program Temporary Fix (PTF) provides a fix to a reported defect. The fix is called temporary because it disappears when it is incorporated into the next release of the product. PTFs may contain a single fix, but generally contain multiple fixes and are associated with a single fileset.

All released PTFs on AIX are classified in order of seriousness:

1 Security advisories

2 Maintenance release information

3 Critical fixes

4 Latest software fixes

5 Installation tips

6 PTFs in errors

7 High impact.

To be informed by e-mail about AIX release correctives, you should subscribe at https://techsupport.services.ibm.com/server/pseries.subscriptionSvcs.

Note: APARs and PTFs are tightly coupled in that PTFs contain multiple APAR fixes. An APAR is a single fix that is delivered via a PTF packaging.

ML

IBM releases cumulative fixes for the AIX operating system that are called Maintenance Level packages. These packages are usually made available two or three times a year for each supported version of AIX.

The Maintenance Level package schedule is published at http://www.ibm.com/servers/eserver/support/pseries/news/2005/01/2005aixschedule.html.

You can subscribe to the Maintenance Bulletins that announce the release of MLs by using the subscription service described above.

Critical Fixes

IBM periodically publishes recommendations for installation of Critical Fixes. These fixes should be used to upgrade systems installed with a particular ML. Fixes identified as critical may also be classified as HIPER, PE, security, or may warrant special awareness. Critical fixes can be downloaded from http://www-1.ibm.com/servers/eserver/support/pseries/criticalfixes.html.

Emergency fixes

In addition to APARs and PTFs, IBM issues emergency fixes. When resolution of a problem cannot wait for a generally-available fix, IBM may provide an emergency fix. The emergency fix (efix) management solution allows users to track and manage efix packages on a system. An efix package might be an emergency fix, debug code, or test code that contains commands, library archive files, or scripts that run when the efix package is installed.

The efix management solution consists of the efix packager (**epkg**) command and the efix manager (**emgr**) command.

The **epkg** command creates efix packages that can be installed by the **emgr** command. The **emgr** command installs, removes, lists, and verifies system efixes.

AIX emergency fixes can be downloaded from http://techsupport.services.ibm.com/server/aix.efixmgmt/home.html.

A convenient collection of URL links pointing to IBM pages enabling manual downloading of various kinds of fixes is located at http://www-1.ibm.com/servers/eserver/support/pseries/aixfixes.html.

COMPARE_REPORT COMMAND

Now you have the ability to identify, find, and download fixes. But how do you actually decide which fixes a particular system needs?

You can use the AIX **compare_report** command to compare the filesets installed on a system with the contents of a fileset image repository or with a list of available updates that may be downloaded from the IBM support site. These comparisons will produce reports that simplify the process of determining the fixes to install to bring a system to the latest maintenance level or the latest level. Reports that are created using the list of available updates can be uploaded directly to http://www-1.ibm.com/servers/eserver/support/pseries/moreservices.html. (Select AIX 5.1, 5.2, or 5.3 to request the exact fixes needed for the system.)

The **compare_report** command was introduced in AIX 5L Version 5.2 and maintenance level 5100-03 of Version 5.1 (APAR IY33992) in the *bos.rte.install* fileset. It provides an easier way to maintain the software installed on the system and encourages updating to the latest maintenance level.

The following example compares the software installed on a system with the list of available updates that can be downloaded from the IBM Web site. It shows the steps to determine the list of fixes necessary to bring the system up to maintenance level 5200-05.

Step 1

Download the file containing a list of available updates from the IBM Web site onto your system.

The file (for example, LatestFixData52) contains a list of fixes that are in the latest maintenance package as well as the latest available fixes that have been released after the latest maintenance package. This file should be on the IBM Web site. A sample portion of the file is shown below:

```
# 2005.03.29
#
# Platform: AIX 5.2
# Data file: LatestFixData52
# Origin: IBM Server Support Site
#
# Description:
# Colon separated file containing a list of what filesets are available
# in the latest maintenance package and all fixes released after the
# latest maintenance package.
#
# Format:
# PTF number : base fileset name : version.release.modification.fix :
package indicator
#
# The package indicator is either the word LATEST_LEVEL or a word
# indicating that the fix is part of the latest maintenance package.
#
U800712:IBMGrid.AppServer:1.1.0.3:LATEST_LEVEL
U800718:IBMGrid.CMM:1.1.0.3:LATEST_LEVEL
U800714:IBMGrid.OGSA:1.1.0.3:LATEST_LEVEL
U800717:IBMGrid.Policy:1.1.0.3:LATEST_LEVEL
U800716:IBMGrid.ServiceGroup:1.1.0.3:LATEST_LEVEL
U800713:IBMGrid.Toolbox:1.1.0.3:LATEST_LEVEL
U800715:IBMGrid.WebApp:1.1.0.3:LATEST_LEVEL
U485984:IMNSearch.rte.DBCS:2.4.0.10:ML520005
U485985:IMNSearch.rte.SBCS:2.4.0.10
…
```

Step 2

Run **compare_report** to generate a report of necessary fixes.

The **compare_report** command is available with AIX Version 5.1 in the 5.1.0.35 or later version of the *bos.rte.install* fileset (APAR IY33992), and in AIX Version 5.2. To run the command on an AIX 5.1 system that is at an earlier version than maintenance level 5100-03, ftp the script for the **compare_report** command.

To determine the fixes necessary to bring a system up to maintenance level 5100-03, compare the software installed on a system (**-s** option) with the report containing a list of available updates (**-r** option). Enter the following command:

```
    /usr/sbin/compare_report -s -r /tmp/LatestFixData51 -l
```

This will generate the two lower-level reports (**-l** option) – */tmp/lowerthanmaint.rpt* (a list of filesets on the system which are at a lower level than the latest maintenance level) and */tmp/lowerthanlatest1.rpt* (list of filesets on the system that are at a lower level than the latest level).

Note: the **compare_report** command in AIX 5.1 is available only from the command line. In AIX 5.2, the command is also available through the SMIT menus using fastpath smitty compare_report.

Sample report output – disk file

A *lowerthanmaint.rpt* file can then be uploaded to the IBM Web site to request the listed PTFs (see Step 3).

Step 3

Upload the comparison report file to the IBM Web site.

The fix release information section of the IBM Web site allows you to upload a file that has been generated by the **compare_report** command. The PTF numbers contained in the comparison report file will be used to provide the requested fixes. After uploading the file, you can download the requested fixes (PTFs) through the normal IBM Web site interfaces.

SUMA – SERVICE UPDATE MANAGEMENT ASSISTANT

The Service Update Management Assistant (SUMA) provides further advancement in the operations required to keep a server or number of servers up to date with the latest maintenance updates. The automation provided by this tool enables a systems administrator to perform the tasks in an unattended and controlled way. In addition, SUMA includes integrated scheduling capabilities to enable repeating tasks and unattended downloads during periods of low network activity. SUMA is installed via the fileset *bos.suma*.

The Service Update Management Assistant is available by default with the AIX 5L Version 5.3 operating system installation. It should be noted that SUMA is available for AIX 5.1 and AIX 5.2 in the form of APARs. All SUMA modules and the **suma** command are contained in the *bos.suma* fileset.

SUMA requires the latest patch level of *perl.rte* (5.6.0.10 for OS Version 5.1; 5.8.0.10 for OS Version 5.2) as well as the *perl.libext* fileset (2.0.56.0 for OS Version 5.1; 2.0.58.0 for OS Version 5.2). These should be included automatically when installing the *bos.suma* fileset.

SUMA also requires an active connection to the Internet. Proxies are supported and secure queries and downloads are supported if OpenSSL is installed. OpenSSL is not installed by default, but is available on the Toolbox for Linux Applications CD.

The **lslpp -p bos.suma** command can be used to verify the requisites for the *bos.suma* fileset:

```
# root:/home/root:  lslpp -p bos.suma
  Fileset               Requisites
  ----------------------------------------------------------------
Path: /usr/lib/objrepos
  bos.suma 5.2.Ø.Ø
 *prereq bos.rte 5.2.Ø.Ø
                        *prereq perl.rte 5.8.Ø.1Ø
                        *prereq perl.libext 2.Ø.58.Ø

Path: /etc/objrepos
  bos.suma 5.2.Ø.Ø
 *prereq bos.rte 5.2.Ø.Ø
                        *prereq perl.rte 5.8.Ø.1Ø
                        *prereq perl.libext 2.Ø.58.Ø
```

Use the **lslpp -f bos.suma** command to examine the list of directories and files that are required to install the SUMA feature:

```
# root:/home/root:  lslpp -f bos.suma
  Fileset               File
  ----------------------------------------------------------------------
Path: /usr/lib/objrepos
  bos.suma 5.2.Ø.Ø
    /usr/suma/lib/SUMA/FixInventory.pm
```

```
                         /usr/suma/lib/SUMA/Policy.pm
               /usr/share/man/info/en_US/a_doc_lib/cmds/aixcmds5/suma.htm
                         /usr/lpp/bos.suma/README
                         /usr/suma/lib/SUMA/Download.pm
                         /usr/suma/lib/msg.map
                         /usr/suma/bin/sm_suma
                         /usr/suma/lib
                         /usr/suma/lib/SUMA/Scheduler.pm
                         /usr/suma/lib/SUMA/DBMStanzaDir.pm
                         /usr/suma/lib/SUMA
                         /usr/suma/lib/SUMA/SoftwareInventory.pm
                         /usr/suma/bin/suma_mgdb
                         /usr/suma/lib/SUMA/NotifyCache.pm
                         /usr/suma/bin/suma
                         /usr/sbin/suma -> /usr/suma/bin/suma
                         /usr/suma/lib/SUMA/PolicyFactory.pm
                         /usr/suma/lib/SUMA/StanzaDB.pm
                         /usr/suma/lib/SUMA/GConfig.pm
                         /usr/suma/lib/SUMA/Util.pm
                         /usr/suma/lib/SUMA/Messenger.pm

Path: /etc/objrepos
  bos.suma 5.2.0.0       /var/suma/tmp
                         /var/suma
                         /var/suma/data
```

SUMA activity is based on the concept of the 'task'. Multiple tasks can be created and stored to perform separate functions. Configurable task defaults are used to fill in underlying options not explicitly overridden when invoking SUMA. Base configuration settings affect all operations.

SUMA functionality is available from the command line via the **suma** command, located in */usr/suma/bin*, with a symbolic link in */usr/sbin*.

The usage information of the **suma** command is shown in the following:

```
root@:/usr/lpp/bos.suma:  suma -h
```

Usage:

- Create, edit, or schedule a SUMA task:

```
   suma { { [-x][-w] } | -s CronSched } [ -a Field=Value ]... [ TaskID ]
```

- List SUMA tasks:

```
suma -l [ TaskID ]...
```

- List or edit the default SUMA task:

```
suma -D [ -a Field=Value ]...
```

- List or edit the SUMA global configuration settings:

```
suma -c [ -a Field=Value ]...
```

- Unschedule a SUMA task:

```
suma -u TaskID
```

- Delete a SUMA task:

```
suma -d TaskID
```

The command can be used to perform the following operations on a SUMA task:

- Create
- Edit
- List
- Schedule
- Unschedule
- Delete.

The specified operation will be performed on the task represented by a unique task identifier (TaskID). If the TaskID is not specified for create or edit operations, the create operation will be assumed, and a unique TaskID will be generated. The **suma -l** command displays all SUMA tasks if the TaskID is not specified. The **suma -c** command entered without any additional flag will list the SUMA global configuration settings.

Each task can implement one of the following actions:

- Preview – specifies that a download preview will be performed. No filesets will be downloaded. In addition to previewing the filesets that would be downloaded, this option can also be used to generate an e-mail notification listing the fixes that are available on the IBM Support Web site.

- Download – specifies that filesets will be downloaded according to the specified request type.
- Download and clean – specifies that filesets will be downloaded based on the specified request type, followed by a clean operation. The clean operation will remove filesets that are not needed from the download target directory, such as updates that have been superseded by a newer fix level that has been downloaded. This can help manage the size of the fix repository.

The request type (RqType) parameter of **suma** commands determines the particular updates to be downloaded.

When **suma** is run with an RqType of Security, Critical, IOServer, or Latest, the RqType is the only required field. Other RqType values (APAR, PTF, ML, Fileset) will require specification of additional Field=Value information:

- APAR – specifies a request to download an APAR.
- PTF – specifies a request to download a PTF.
- ML – specifies a request to download a specific maintenance level.
- Fileset – specifies a request to download a specific fileset.
- Security – specifies a request to download the latest security fixes.
- Critical – specifies a request to download the latest critical fixes.
- IOServer – specifies a request to download the latest I/O server fixes.
- Latest – specifies a request to download all the latest fixes.

SUMA performs various types of filtering on the fixes requested for download. To ensure that only the desired fixes will be downloaded, comparisons can be performed against an inventory of software installed on a system, a directory containing a repository of fixes, or the maintenance level of the system .

At the completion of a download operation, SUMA displays logs, or e-mails a summary report showing the number of fixes that were downloaded successfully, any fixes that failed to download, and fixes that were skipped because they were already present on the system. A preview option is also available if you wish to generate a list of fixes that are available on the IBM Support Web site.

SUMA downloads can be scheduled or executed immediately. By scheduling a policy, an unattended download can be set up to conform to a client's maintenance window. SUMA also provides support for scheduling policies at flexible intervals, such as hourly, daily, weekly, and monthly. This allows the download to occur at an optimal time or can be used to establish the frequency to check for the availability of certain fixes.

Scheduled policies can also be designated as repeating. Repeating policies for certain fix types, such as APAR, PTF, or Maintenance Level, will be repeated according to the selected interval, and then deleted when the specific fix or maintenance level is found. Other fix types, those released as a group on an on-going basis, such as Critical, Security, I/O Server and Latest, will continually be repeated at the specified interval until the policy is unscheduled. Policies can also be scheduled to be non-repeating, in which case they will be run once and then removed from the system.

Another area where SUMA provides a lot of flexibility is logging. SUMA supports six different verbosity levels that can be uniquely set for sending information to the screen, a log file, or as a part of an e-mail notification. The levels are listed below, ordered from the least to the most amount of information displayed:

- OFF – no information is displayed or logged.
- ERROR – displays error messages and other highly important messages.
- WARNING – displays warning messages in addition to ERROR messages.
- INFO – displays informational messages in addition to WARNING messages.
- VERBOSE – displays verbose informational messages in addition to INFO messages.
- DEBUG – displays debug output for debugging purposes. This setting is not used for normal operations.

To enable your system for automated downloads of maintenance fixes, SUMA provides both command line and menu-driven System Management Interface Tool (SMIT) interfaces.

REFERENCES

1 *AIX 5L Differences Guide Version 5.3 Edition*, SG24-7463-00.

2 Update your AIX system with SUMA – http://www-128.ibm.com/developerworks/eserver/library/es-updateaix.

3 Suma Command Reference – http://publib.boulder.ibm.com/infocenter/pseries/index.jsp?topic=/com.ibm.aix.doc/cmds/aixcmds5/suma.htm.

Alex Polak
System Engineer
APS (Israel)

Run a common script across a list of servers

INTRODUCTION

This script, runbatch, is used to run a common script across a list of one or more servers. The script uses passwordless ssh to execute the script on remote servers, and distribution of the script is done through secure copy (scp). Only new versions of the common script will be remotely copied if the RCS (Revision Control System) version number is different from the local version.

Typical usage would be to execute a script on all your servers and collect the output in a log file. An error log file is also kept.

Note: this script can easily be modified to work on any flavour of Unix by simply setting the appropriate PATH variable to match the system commands of your Unix system.

PREREQUISITES

OpenSSH V3.6p1 (or more recent) is prerequisite software that needs to be installed on every server.

RCS (Revision Control System) is prerequisite software that needs to be installed on at least one server.

The RCS software can be found and downloaded from IBM's AIX Toolbox Download page at http://www-1.ibm.com/servers/aix/products/aixos/linux/download.html.

All software found on IBM's AIX Toolbox Download page is in RPM (Redhat Package Manager) format. So, you will need to have the *rpm.rte* LPP installed before you can install the RPMs. The *rpm.rte* AIX installp format can be found at ftp://ftp.software.ibm.com/aix/freeSoftware/aixtoolbox/INSTALLP/ppc/rpm.rte.

OpenSSH can be found on the Bull Freeware site at http://www.bullfreeware.com.

SOFTWARE INSTALLATION

Installation of Revision Control System

Revision Control System is a file version management tool. You can download RCS from IBM's AIX Toolbox Download page in RPM format.

To install the package, simply type the command:

```
# rpm -i rcs-5.7-2.aix5.1.ppc.rpm
```

Installation of openSSH

OpenSSH is the open-source version of secure shell. You can download it from the Bull Freeware site (www.bullfreeware.com) for any version of AIX. To install the package, which comes in exe format, simply follow the instructions found on Bull's site at www.bullfreeware.com/install_down.html.

Once the package is installed, you will have to generate the server's keys by using the command:

```
# ssh-keygen -t rsa1 -f /usr/local/ssh/etc/ssh_host_key -N  ""
# ssh-keygen -t rsa -f /usr/local/ssh/etc/ssh_host_rsa_key -N  ""
# ssh-keygen -t dsa -f /usr/local/ssh/etc/ssh_host_dsa_key -N  ""
```

You may have to change the path where your server's keys will be stored (ie */usr/local/ssh/etc*).

SETTING UP PASSWORDLESS SSH

In order to set up passwordless ssh, you must first create a user on all your servers that will be used to execute the common script. You may optionally use an existing user or even the root account (at your risk).

For my example, I chose to create a special user called 'batch' and group 'batch' to execute this script.

First create a new AIX group, which will contain only the passwordless SSH user, batch. I have chosen to name the group batch with a gid of 522:

```
# mkgroup -'A' id='522' batch
```

Next, create the user batch with a uid of 673 and a shell of ksh:

```
# mkuser id='673' pgrp='batch' groups='batch' shell='/usr/bin/ksh'
home='/home/batch' \
gecos='Runbatch script user' batch
```

Note: do not give this user a password in AIX. Authentication will be done using SSH.

In order to allow the user batch to log in to all servers without a password, you must generate the private/public key pair.

First, as root, change user to batch, and in batch's home directory type:

```
# su - batch
# ssh-keygen -t dsa -f .ssh/id_dsa
```

A password will be asked for. Do not enter a password, just press *Enter*.

Now, go the *.ssh* directory, and you will find two new files – *id_dsa* and *id_dsa.pub*.

The second one is the public key. Rename it thus:

```
# cd .ssh
# mv id_dsa.pub authorized_keys2
# chmod 600 authorized_keys2
```

Now, try to log in to the same server using ssh to verify that things have worked.

Note: the first time you log in to a server using ssh, even though it is passwordless, it will ask you if you want to permanently add that host to the list of known hosts. You must answer 'yes'.

The next time you log in to the remote server, no password or questions will be asked.

Note: this system will work as long as none of the machines changes its IP address. If this server changes its IP address, you will have to regenerate the public/private keys. If any of the other servers changes its IP address, you will have to manually delete it from the *known_hosts* file in the *.ssh* directory and answer ‘yes’ to permanently add that host to the list of known hosts again.

You will have to run this procedure on every server that you wish to be able to run the common script. However, you will not need to recreate the private/public key pair on every server.

After you have created the group and user on your server, simply copy the file from the first server onto the new server:

```
/home/batch/.ssh/authorized_keys
```

Now, change the permissions for the local batch account thus:

```
# chown -R batch:batch /home/batch
# chmod -R 700 /home/batch
# chmod 600 /home/batch/.ssh/authorized_keys
```

Once again, try to log in to this new server using ssh to verify that things worked correctly.

Continue this procedure until all your servers have been configured.

SETTING UP VERSIONING USING RCS

In order to ensure that we do not needlessly copy the common script to every server, we will use RCS to version our scripts. This practice also allows us to keep back-ups of our scripts.

For help on using RCS, see http://www.gnu.org/software/rcs/rcs.html.

For the purpose of this article, I will show you some basic RCS commands to set up versioning of our scripts.

First, create a directory named RCS, which will hold RCS version information:

```
# mkdir /home/batch/RCS
```

Now, in your common scripts (my test script is called *script.sh*), do the following:

1 Add $Id$ at the top of the script:

```
#
# $Id$
#
```

2 Check-in (ci) the file:

```
# ci -i script.sh
```

3 Check-out and check-in the file again:

```
# co -l script.sh
# ci -u script.sh
```

4 Look in file *script.sh* now:

```
#
# $Id: script.sh,v 1.1 2005/02/24 17:23:58 prattico Exp $
#
```

Now, every time you wish to make a change to your script, you simply follow this sequence:

1 Check-out the file using the **co** command:

```
co -l script.sh
```

2 Modify the script:

```
vi script.sh
```

3 Check-in the file using the **ci** command:

```
ci -u script.sh
```

Your version number will automatically increase at every check-in if there is a change to the script.

RUNBATCH SCRIPT

```
#!/bin/sh
# Set your PATH variable for system commands here
export PATH=/usr/bin:/usr/sbin
# Some script name and server settings
BASENAME='basename $0'                              # Script name
HOSTNAME='hostname'                         # Where am I running
###################################################################
# Modify these parameters to match your environment
###################################################################
SERVERLIST=/home/batch/serverlist
                           # Path to file with all your server names
```

```
ERRFILE=$BASENAME"_error.log"
                             # Default file name where errors will go
LOGFILE=$BASENAME"_results.log"
                    # Default file name where scripts results will go
USER=batch            # User which can ssh to all server passwordless
SSH=/usr/local/bin/ssh                  # Path to ssh executable
SCP=/usr/local/ssh/bin/scp              # Path to scp executable
######################################################################
# DO NOT MODIFY ANYTHING BELOW HERE UNLESS YOU KNOW WHAT YOU'RE DOING
######################################################################
# Initialize some variables
SERVERS=""
SCRIPT=""
VERBOSE=0
BREAK=0
# Help on script usage function
function usage {
   echo "\n   The script $BASENAME is used to execute a script"
   echo   "   on one or more AIX servers using passwordless ssh."
   echo "\n   Syntax: $BASENAME -b script [-s \"host1 host2...
hostn\"][-v][-o outputfile][-e errorfile][-h]"
   echo "\n   Description of flags and parameters:"
   echo "\n       -h : help"
   echo   "       -b : full path to script to execute"
   echo   "       -s : list of servers to process"
   echo   "       -v : verbose mode"
   echo   "       -o : Path to file which will store results (default:
$LOGFILE)"
   echo   "       -e : Path to file which will store errors (default:
$ERRFILE)"
   echo "\n       Examples: $BASENAME -b script.sh"
   echo   "                 $BASENAME -b script.sh -s \"myserv1
myserv2\""
   echo   "                 $BASENAME -h\n\n"
   exit
}
# Preliminary checks function - basic sanity checks
function run_preliminary_checks {
      if [[ ! -r $SERVERLIST  && -z $SERVERS ]];
      then
            echo "\nERROR: file $SERVERLIST is not available\n"
            exit 1
      fi
      lsuser $USER > /dev/null 2>&1
      if [[ $? != 0 ]];
      then
            echo "\nERROR: User $USER does not exist\n"
            exit 1
      fi
          if [[ ! -x $SSH ]];
          then
                echo "\nERROR: $SSH is not available\n"
                exit 1
          fi
          if [[ ! -x $SCP ]];
          then
                echo "\nERROR: $SCP is not avaialable\n"
                exit 1
          fi
}
# Function to find local version of script to execute on all servers
function get_local_version {
      LV='grep "\\$Id" ${SCRIPT} | grep -v grep | awk '{print $4}''
      if [[ -z $LV ]];
      then
            echo "\nERROR: Version of script $SCRIPT on $HOSTNAME is not
available\n"
            exit 1
      fi
      [ $VERBOSE -eq 1 ]  &&  echo "Local version of script : $SCRIPT is
$LV\n"
}
# Function to find remote version of script to execute on all servers
function check_version {
      # $1=SERVER $2=SCRIPT
      CMD="su - $USER -c $SSH $1 grep -s '\\\$Id' $2 | grep -v grep |
awk '{print \$4}'"
      RV='$CMD'
      if [[ -z $RV ]];
      then
            [ $VERBOSE -eq 1 ]  &&  echo "WARNING: Script $2 on $1 is
not available\n"
      else
            [ $VERBOSE -eq 1 ]  &&  echo "Remote version of script : $2
is $RV\n"
      fi
}
# Function which executes script on remote server
function run_script {
      # $1=SERVER $2=SCRIPT
      [ $VERBOSE -eq 1 ]  &&  echo "Running script $2 on server $1\n"
      su - $USER -c $SSH -q $1 $2 >> $LOGFILE
}
# Function to copy script to remote server using scp
function scp_latest_version {
      # $1=SERVER $2=SCRIPT
      su - $USER -c $SCP -q $2 $USER@$1:$2
}
# Function to verify ssh is available and functional
function check_ssh {
      su - $USER -c $SSH -q -l $USER $1 "exit";
      if [[ $? -ne "Ø" ]];
      then
            echo "\nERROR: ssh for user $USER is not functional on
server $1" >> $ERRFILE
            BREAK=1;
      fi
}

# Main
while getopts :b:s:o:e:hv arg
do
        case $arg in
      h)      usage
                ;;
      b)      SCRIPT=$OPTARG
            ;;
      s)      SERVERS=$OPTARG
            ;;
      o)      LOGFILE=$OPTARG
            ;;
      e)      ERRFILE=$OPTARG
            ;;
              v)      VERBOSE=1
                ;;
              \?)      usage
                ;;
              *)      usage
                ;;
        esac
done
# Verify we have the name of the script to execute on servers
if [[ -z $SCRIPT ]];
then
      echo "\nERROR: the path to the script to execute is required!\n"
      usage
fi
# Script exists and is it executable?
if [[ ! -x $SCRIPT ]];
then
      echo "\nERROR: the script $SCRIPT is not available or not
executable!\n"
      echo "Verify, correct and try again.\n"
      exit
fi
# Erase old work files, in case...
/usr/bin/rm -f $LOGFILE $ERRFILE
run_preliminary_checks
get_local_version
if [[ -z $SERVERS ]];
```

```
then
      for server in 'cat $SERVERLIST'
      do
    [ $VERBOSE -eq 1 ]  &&  echo "Processing server $server - `date`..."
            check_ssh $server
            if [[ $BREAK -eq 1 ]];
            then
                  BREAK=0;
                  continue
            fi
            check_version $server $SCRIPT
            if [[ $LV != $RV ]];
            then
                  [ $VERBOSE -eq 1 ]  &&  echo "Synchronising $SCRIPT on
remote server $server"
                  scp_latest_version $server $SCRIPT
            fi
            run_script $server $SCRIPT
      done
else
      for server in $SERVERS
      do
    [ $VERBOSE -eq 1 ]  &&  echo "Processing server $server - `date`..."
            check_ssh $server
            if [[ $BREAK -eq 1 ]];
            then
                  BREAK=0;
                  continue
            fi
            check_version $server $SCRIPT
            if [[ $LV != $RV ]];
            then
                  [ $VERBOSE -eq 1 ]  &&  echo "Synchronising $SCRIPT on
remote server $server"
                  scp_latest_version $server $SCRIPT
            fi
            run_script $server $SCRIPT
      done
fi
[ $VERBOSE -eq 1 ]  && echo "\nScript output file : $LOGFILE"
[ $VERBOSE -eq 1 ]  && echo "Script error file  : $ERRFILE\n"
exit 0
```

TESTING

In order to test that everything is working properly, create a test script that issues a simple command such as **hostname**. Make sure to use RCS.

Here is my sample script:

```
#!/bin/sh
# $Id: script.sh,v 1.1 2005/02/24 17:23:58 prattico Exp $
#
/usr/bin/hostname
```

Now, issue the runbatch script as follows:

```
# runbatch -b script.sh -s "server1 server2 server3" -v
```

Verify your output logs to ensure that everything ran smoothly.

CREATING A SERVERLIST FILE

If you want to simplify batch execution, the runbatch script can optionally use a file with a list of servers on which to run the common script.

The format of the serverlist file is a single server name per line with no blank lines allowed in the text file. Here is an example:

```
server1
server2
server3
servern
```

The default path for your serverlist file is */home/batch/serverlist*. If you change the path, simply make the same change in the runbatch script.

To use the serverlist file, do not put the **-s** flag on the command line:

```
# runbatch -b script.sh -v
```

Elvio Prattico
Consultant
PRATTICO Consulting (Canada)

Tape management system – part 2

This month we continue the code for a tape management system.

DB_UPDATE.KSH

```
#!/bin/ksh
#####################################################################
# This script will update the tape database.  It should not     #
# be executed by a user.                                         #
#####################################################################
home=/var/tapesys
host='hostname'
pid="$$"
Date='date'
label_log="$home/bin/label_log.ksh"
# Check freespace in database directory
df -k $home|tail -1|read a b free d e
$label_log "Database freespace: $free kB"
if [ "$free" -lt "10240" ] ; then
   $label_log "Full filesystem - unable to update database"
   echo "Error updating tape database - directory full"
   echo "Call AIX Support.  Process will pause here."
echo "\n`tput smso`DO NOT Press [ENTER] until told to do so.`tput rmso`"
   read foo
   echo "Safety check - Press [ENTER] again."
   read anotherfoo
fi
```

$HOME/BIN/BACKUP.KSH

```
$home/bin/lockdb.ksh "DB Update"
   date='date'
   sec="$home/bin/sec"
   s2d="$home/bin/s2d"
   no_rewind=".1"
   args="$#"
   my_fullname="$0"
   myname='basename $0'
   tmp="/tmp/tapedb.$$"
#  tapes.db is a listing of the most current versions of each tape.
#  tape_history.db is the status of all tapes since they have been used.
   if [ "$args" -eq "2" ] ; then
      device="$1"
      backup="$2"
      mt -f $device rewind
      $label_log "Read tape label from db_update"
      label='$home/bin/tape_label.ksh get $device'
      echo $label:$host:$backup:Ø >>$home/db/tape_history.db
      volser='echo "$label"|cut -f1 -d":"'
      $label_log "Begin module 1 update tape database for volser:
$label:$host:$backup"
      $label_log "Tape history updated from module 1"
      grep -v $volser $home/db/tapes.db >$tmp
      echo $label:$host:$backup:Ø >>$tmp
      $label_log "Create temporary volser record from module 1"
      $label_log "End module 1 update tape database for volser:
$label:$host:$backup"
   else
      $label_log "Begin module 2 update tape database for volser: $1"
      echo "$1" >>$home/db/tape_history.db
      $label_log "Tape history updated from module 2"
      volser='echo "$1"|cut -f1 -d":"'
      $label_log "Module 2 update tape database for volser: $volser"
      grep -v $volser $home/db/tapes.db >$tmp
      $label_log "Database updated from module 2"
      echo $1 >>$tmp
      $label_log "End module 2 update tape database for volser: $1"
   fi
   cp -p $tmp $home/db/tapes.db
   $label_log "Database updated from main module"
   rm $tmp
$home/bin/unlockdb.ksh "DB Update"
```

DEFCHECK.KSH

```
#!/bin/ksh
# Script to check the tape database to see whether a currently
# mounted tape is defective or not.
tape_drive=${1}
home="/var/tapesys"
### tmp="/tmp/tape.$$"
tapes="$home/db/tapes.db"
label_log="$home/bin/label_log.ksh"
host='hostname'
rc=Ø
while : ; do
   mt -f $tape_drive rewind 2>/dev/null
   if [ "$?" -ne "Ø" ] ; then
      echo "`basename $Ø`: Tape drive $tape_drive is not ready"
      echo "or other tape error."
      while : ; do
         echo " "
         echo "Press  a  to abort this command"
         echo "Press  c  to continue this command immediately"
         echo "Press  w  to wait 2Ø seconds and try again"
         echo " "
         echo "Enter here   --> \c"
         read option
         if [ -z "$option" ] ; then
            echo "Please select a/c/w"
            continue
         fi
         if [ "$option" = "a" ] ; then
            rc=2
            exit $rc
         fi
         if [ "$option" = "c" ] ; then
            break
         fi
         if [ "$option" = "w" ] ; then
            sleep 2Ø
            break
         fi
         continue
      done
   else
      break
   fi
done
$label_log "Defective tape check"
### $home/bin/tape_label.ksh get $tape_drive >$tmp
tmp_label=$($home/bin/tape_label.ksh get $tape_drive)
### volser=$(echo ${tmp_label}|cut -f1 -d":")
volser=$(echo ${tmp_label}|cut -f1 -d":")
### created=$(echo ${tmp_label}|cut -f2 -d":")
created=$(echo ${tmp_label}|cut -f2 -d":")
foo='grep $volser $tapes|grep -i defective'
defective_tape="$?"
# If the result code is "Ø", then we found the string
# "Defective" in the database.
### rm $tmp
if [ "$defective_tape" -eq "Ø" ] ; then
   $label_log "volser $volser is defective"
   exit 1
else
   $label_log "volser $volser is NOT defective"
   exit Ø
fi
# Exit with a "1" if the tape is defective, and "Ø" if it is not.
```

LABEL_LOG.KSH

```
#!/usr/bin/ksh
# This program tracks activity as tape labels are read/written
home="/var/tapesys"
h=$(hostname)
label_log="${home}/log/tapesyslog"
argv="$@"
d=$(date)
echo "$d $h: $argv" >> $label_log
exit Ø
```

LOCKDB.KSH

```
#!/usr/bin/ksh
home=/var/tapesys
host='hostname'
pid="$$"
lock="$home/db/dbupdate.lck"
Args="$@"
Date='date'
label_log="$home/bin/label_log.ksh"
image="%-1Ø.1Øs%-6Ø.6Øs%-25.25s\n"
if [ -r $lock ] ; then
   printf $image "$host:" "database lock attempt failed $Date" "($Args)"
>> $home/log/dblocklog.log
   $label_log "Database lock failed"
   while : ; do
      echo "\n\nThe tape database is currently being updated"
      cat $lock
      echo "I will wait until the other process finishes"
      echo "To clear this condition, remove the file $lock."
      echo "But be sure you know what you are doing..."
      sleep 5
      if [ ! -r $lock ] ; then
           echo "\nLock has been released...continuing\n\n"
        tput rmso
           break
      else
           echo "Lock is still present"
      fi
   done
fi
echo "Host: $host    Process: $pid     Date: $Date" >$lock
$label_log "Database locked"
printf $image "$host:" "database locked $Date" "($Args)" >> $home/log/
dblocklog.log
```

PRINTDF.KSH

```
#!/bin/ksh
# Menu for generating reports on the tape system
# Written 3/2/2001 Bill V.
z=0
y=0
x=0
w=0
format="%-35.30s%-15.12s%-15.12s%-8.6s%-10.8s%-8.6s%-35.33s\n"
# Filesystem    1024-blocks      Free %Used    Iused %Iused Mounted on
# /dev/exportlv     2949120   2171708   27%      708     1% /furn_export
df -k |head -1|read a b c d e f g h i
printf $format "$a" "$b" "$c" "$d" "$e" "$f" "$g $h"
for line in 'lsfs|awk {'print $3'}' ; do
   df -k $line 2>/dev/null |tail -1|read a b c d e f g h i
   if [ "$?" -eq "1" ] ; then
      continue
   fi
   mount|grep $line 2>/dev/null 1>&2
   if [ "$?" -eq "1" ] ; then
      continue
   fi
   printf $format "$a" "$b" "$c" "$d" "$e" "$f" "$g $h"
   z='expr $z + 1'
   y='expr $y + $b'
   x='expr $x + $c'
done
echo ""
printf $format "Total: $z" "$y" "$x"
exit 0
```

PRINTL.KSH

```
#!/bin/ksh
# Script wrapper to print out stuff!
if [ "$#" -ne "2" ] ; then
  echo "Oops - try again"
  echo "$0 queue file"
  exit 1
fi
q=$1
f=$2
date='date +%m/%d/%Y'
time='date +%H:%M:%S'
date_time="$date $time"
title='### TAPE REPORTS ###'
tmp="/tmp/printl.$$"
# echo $date_time >$tmp
echo $title >> $tmp
```

```
cat $f >>$tmp
qprt -dc -#v -#j -P $q -q '6ØØ' -s lineprinter -p17 -T "$title2" -z'+'
$tmp 2>/dev/null
if [ "$?" -ne "Ø" ] ; then
   echo "There was a problem printing the report."
   echo "I will try again with different parameters,"
   echo "however, the report may be more difficult to read.\n"
   lp -d $q $f 2>/dev/null
   sleep 1Ø
fi
sleep 5
rm $tmp
```

PRINTP.KSH

```
#!/bin/ksh
# Script wrapper to print out stuff!
if [ "$#" -ne "2" ] ; then
  echo "Oops - try again"
  echo "`basename $Ø` queue file"
  exit 1
fi
q=$1
f=$2
date='date +%m/%d/%Y'
time='date +%H:%M:%S'
date_time="$date $time"
title='### TAPE REPORTS ###'
tmp="/tmp/printl.$$"
# echo $date_time >$tmp
echo $title >> $tmp
cat $f >>$tmp
qprt -dc -#v -#j -P $q -q '6ØØ' -T "$title2" $tmp 2>/dev/null
if [ "$?" -ne "Ø" ] ; then
   echo "There was a problem printing the report."
   echo "I will try again with different parameters,"
   echo "however, the report may be more difficult to read.\n"
   lp -d $q $f 2>/dev/null
   sleep 1Ø
fi
sleep 5
rm $tmp
```

PROC_HIST_REPORT.KSH

```
#!/usr/bin/ksh
VOLSER="$1"
home="/var/tapesys"
s2d="/$home/bin/s2d"
```

```
tape_history="$home/db/tape_history.db"
ofile="$home/reports/tape_detail.$VOLSER"
>$ofile
format="%-13s%-29.26s%-29.26s%-10.10s%-5s%-40.38s\n"
for tape_rec in 'cat $tape_history|grep ^$VOLSER' ; do
   volser='echo $tape_rec|cut -f1 -d":"'
   if [ "$pvol" = " " ] ; then
      pvol=$volser
   fi
   creation='echo $tape_rec|cut -f2 -d":"'
   lifetime='echo $tape_rec|cut -f3 -d":"'
   host='echo $tape_rec|cut -f4 -d":"'
   backup='echo $tape_rec|cut -f5 -d":"'
   location='echo $tape_rec|cut -f6 -d":"'
   creation_time='$s2d $creation'
   foo='echo "$creation+$lifetime"|bc'
   expiration_time='$s2d $foo'
# ###    passes='grep $volser $tape_history|wc -l'
   if [ "$location" -eq "0" ] ; then
      loc="ON"
   elif [ "$location" -eq "1" ] ; then
      loc="OFF"
   else
      loc="UNK"
   fi
   printf $format "$volser" "$creation_time" "$expiration_time" "$host"
"$loc" "$backup" >> $ofile
   passes='grep $volser $tape_history|wc -l'
   echo "Total: $passes" >> $ofile
done
```

PROC_RANGE.KSH

```
#!/bin/ksh
# Receive input in the form of "10-15,6,8,20-30,4-5" and it will
# return:
# 10 11 12 13 14 15 6 8 20 21 22 23 24 25 26 27 28 29 30 4 5
# This should be fun.
quiet=0
nl=1
if [ "$#" -ne "0" ] ; then
   while [ "$#" -gt "0" ] ; do
      var=$1
      if [ "$var" = "-q" ] ; then
         quiet=1
      fi
      if [ "$var" = "-1" ] ; then
         nl=0
      fi
      shift
   done
fi
if [ "$quiet" -eq "Ø" ] ; then
   echo "Enter a range of numbers, format a-b,c,d-e ..."
fi
read range
orig_range="$range"
done=Ø
delim=","
field=Ø
terminator=",EOD"
range1="${range}${terminator}"
range=$range1
if [ "$quiet" -eq "Ø" ] ; then
   echo "Range is $orig_range:\n"
fi
while [ $done -eq Ø ] ; do
   field='expr $field + 1'
   data='echo "$range"|cut -f $field -d "$delim"'
   if [ "$data" = "" ] ; then
#    No commas found, assume only one range
     data="$range"
   fi
   if [ "$data" = "EOD" ] ; then
      done=1
      break
   fi
   foo='echo "$data"|grep "-" 1>/dev/null'
   if [ "$?" -eq "Ø" ] ; then
      left='echo "$data"|cut -f1 -d"-"'
      right='echo "$data"|cut -f2 -d"-"'
      while [ "$left" -le "$right" ] ; do
        echo "$left \c"
        if [ "$nl" -eq "1" ] ; then
           echo ""
        fi
        left='expr $left + 1'
      done
   else
      echo "$data \c"
      if [ "$nl" -eq "1" ] ; then
        echo ""
      fi
   fi
done
if [ "$nl" -eq "Ø" ] ; then
   echo " "
fi
```

SCRATCH_REPORT.KSH

```
#!/usr/bin/ksh
home="/var/tapesys"
label_log="$home/bin/label_log.ksh"
$label_log "Running scratch report"
echo "/dev/null" | $home/bin/tape_report.ksh foo scratch ### 1>/dev/null
2>&1
# ### DT='date +%D-%T'
# ### cat $home/reports/scratchreport.txt|mail -s "Tape Scratch Report
for $DT" ops,root
```

SCRATCH_TEST.KSH

```
#!/bin/ksh
trap "exit 1" 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
#set -x
tape_drive="${1}"
home="/var/tapesys"
# It would seem that some other processes sometimes
# cause a device contention when doing a scratch test.
# Perhaps this is caused by a previous command not finishing
# with the tape driver (it is still perhaps rewinding).
# So, I'll put a sleep here....
sleep 15
h=$(hostname)
label_log="${home}/bin/label_log.ksh"
retries=3
retried=1
while : ; do
   ${label_log} "Scratch test"
   tmplabel=
   tmplabel=$(${home}/bin/tape_label.ksh get ${tape_drive})
   volser=$(echo ${tmplabel}|cut -f1 -d":")
   VL=$(echo "${volser}" |wc -c |awk {'print $1'})
   if [ "$VL" -lt "6" ] ; then
      while [ "$retried" -lt "$retries" ] ; do
         $label_log "Corrupted VOLSER $volser detected ${retried}"
         echo "Attention - corrupted tape information received"
         echo "Attempt number ${retried} to recover."
         sleep 15
         mt -f ${tape_drive} rewind
         sleep 15
         ${label_log} "Scratch test retry"
         tmplabel=
         tmplabel=$(${home}/bin/tape_label.ksh get ${tape_drive})
         volser=$(echo ${tmplabel}|cut -f1 -d":")
         VL=$(echo "${volser}" |wc -c |awk {'print $1'})
         if [ "$VL" -lt "6" ] ; then
            retried='expr $retried + 1'
```

```
           else
             echo "Successfully recovered tape label on attempt ${retried}."
       $label_log "Successfully recovered tape label on attempt ${retried}."
              $label_log "Offending tape is: $tmplabel"
              retried=99
              break
           fi
           done
             if [ "$retried" -ne "99" ] ; then
              echo "After ${retried} attempts, unable to read tape label"
                echo "Contact AIX on-call support ASAP"
                echo "Press [ENTER] to exit after you have called someone"
                echo "and you have been instructed what to do next."
              read foo
                exit 1
           fi
       fi
       created=$(echo ${tmplabel}|cut -f2 -d":")
       retention=$(echo ${tmplabel}|cut -f3 -d":")
       if ${home}/bin/synch_test.ksh $tmplabel
       then
          ${label_log} "volser ${volser} is in synch with db"
          synch_test=0
       else
          ${label_log} "volser ${volser} is NOT in synch with db"
          synch_test=1
       fi
       now='$home/bin/sec'
       ((scratch = ${created} + ${retention}))
       scratch_date='$home/bin/s2d $scratch'
       now_date='$home/bin/s2d $now'
       if [ "${scratch}" -le "$now" ] ; then
          scratch_tape=1
       else
          scratch_tape=0
       fi
       sleep 15
       if [ "${synch_test}" -eq "1" ]
       then
          echo "Tape $volser is not in synch with the database  - I'll try
   the next one"
          echo "[Press Ctrl-C to cancel this operation at any time]"
          $label_log "volser $volser is NOT in synch with the database"
          mt -f $tape_drive rewoffl
          if [ "${h}" = "sapqam" -o "${h}" = "discus1" ] ; then
                sleep 90  # Give the library time to switch tapes.
          else
                echo "Press [ENTER] after a new tape has been inserted"
                read somethingnew
          fi
       elif [ "${scratch_tape}" -eq "Ø" ]
       then
          echo "Tape $volser is not scratch - I'll try the next one"
          echo "[Press Ctrl-C to cancel this operation at any time]"
          $label_log "volser $volser is NOT scratch"
          mt -f $tape_drive rewoffl
          if [ "${h}" = "sapqam" -o "${h}" = "discus1" ] ; then
                sleep 9Ø  # Give the library time to switch tapes.
          else
                echo "Press [ENTER] after a new tape has been inserted"
                read somethingnew
          fi
       else [ "$scratch_tape" -eq "1" ]
          $label_log "volser $volser is scratch and in synch with database"
          tmplabel=
          exit Ø
       fi
    done
```

YNC.KSH

```
#!/bin/ksh
# This is a report that will read tape labels and compare
# the internal tape library with the tape database.
# Exceptions will be identified in the report output.
# Command line options:
# $1 = tape device
if [ "$#" -ne "1" ] ; then
   echo "Please specify a tape drive...\n"
   exit 1
fi
device=$1
boj='date'
home="/var/tapesys"
tapes="$home/db/tapes.db"
Ymd='date +%Y%m%d'
hms='date +%H%M%S'
report="$home/reports/syncreport.${Ymd}_${hms}.txt"
tmp="/tmp/sync.$$.txt"
i1="%-72.5Øs%-5Ø.4Øs\n"
i2="%-1Ø.8s%-31s%-31s%-1Ø.8s%-31s%-34.31s%-2s\n"
touch $report
printf $i1 "Tape Information" "Library Information" >>$report
printf $i2 "Volser" "Creation Date/Time" "Expiration Date/Time" "Volser"
"Creation Date/Time" "Expiration Date/Time" " " >>$report
clear
>$tmp
while : ; do
   t=Ø
   x=1
   echo "How many tapes are in the library?"
   echo "Enter \"Ø\" to finish reading and"
   echo "generate the report"
   read t
   if [ "$t" -ne "Ø" ] ; then
      while [ "$x" -le "$t" ] ; do
         dd if=$device count=1 2>/dev/null |tee -a $tmp
         mt -f $device rewoffl
        echo "Press [ENTER]"
         read foo
         x='expr $x + 1'
      done
   else
      break
   fi
done
clear
echo "Processing report..."
for line in 'cat $tmp' ; do
   tvolser='echo "$line"|cut -f1 -d":"'
   tcreate='echo "$line"|cut -f2 -d":"'
   tscratc='echo "$line"|cut -f3 -d":"'
   t1='expr $tcreate + $tscratc'
   lvol='grep "$tvolser:" $tapes'
   lvolser='echo "$lvol"|cut -f1 -d":"'
   lcreate='echo "$lvol"|cut -f2 -d":"'
   lscratc='echo "$lvol"|cut -f3 -d":"'
   l1='expr $lcreate + $lscratc'
   tc='$home/bin/s2d $tcreate'
   ts='$home/bin/s2d $t1'
   lc='$home/bin/s2d $lcreate'
   ls='$home/bin/s2d $l1'
   if [ "$tcreate" -ne "$lcreate" -o "$t1" -ne "$l1" ] ; then
      flag="X"
   else
      flag=" "
   fi
   printf $i2 "$tvolser" "$tc" "$ts" "$lvolser" "$lc" "$ls" "$flag" >>
$report
done
echo "Report is $report"
```

SYNCH_TEST.KSH

```
#!/bin/ksh
trap "exit 1" 1 2 3 4 5 6 7 8 9 1Ø 11 12 13 14 15
#set -x
### current tapesys location
home="/var/tapesys"
```

```
tapes_db="${home}/db/tapes.db"
### hostname should include vars.ksh but would need to override home
dir.  Will change soon.
h=$(hostname)
### script for logging.
label_log="${home}/bin/label_log.ksh"
### Log synch test
${label_log} "Synch test"
### the label from the tape should be passed as an argument.  That way
we don't need to read the tape again.
tmplabel=${1}
### get volser from the tape label so we can compare to database
volser=$(echo ${tmplabel}|cut -f1 -d":")
### search tape database for volser.
db_label=$(grep ${volser} ${tapes_db})
### Set database label to be the first 3 fields to compare to tapelabel.
db_label="$(echo ${db_label}|cut -f1 -d":"):$(echo ${db_label}|cut -f2 -
d":"):$(echo ${db_label}|cut -f3 -d":")"
### compare labels from tape and database.  return 0 if it's the same, 1
otherwise
### echo "dblabel: ${db_label} \ntmplabel: ${tmplabel}"
if [ ${db_label} != ${tmplabel} ]
then
      exit 1
else
      exit 0
fi
```

TAPE_LABEL.KSH

```
#!/bin/ksh
#####################################################################
# This script will be the wrapper around our tape management      #
# system.  You pass this script the following parameters:         #
# tape_label.ksh [action] [tape device]                           #
#    Action:                                                      #
#                  get                 Reads the tape label       #
#                  init                Initialize new tapes       #
#                  make [retention]   Writes a tape label         #
#                       with a retention = [retention]            #
#                  change              Change a tape label        #
#                                                                 #
#                  tape device    A tape drive in /dev/???        #
#                                 format                          #
# Example: tape_label.ksh make /dev/rmt2                          #
#          Creates a tape label on the tape in /dev/rmt2          #
#####################################################################
#date=$(date)
home="/var/tapesys"
```

```
label_log="${home}/bin/label_log.ksh"
sec="${home}/bin/sec"
s2d="${home}/bin/s2d"
host=$(hostname)
no_rewind=".1"
args="$#"
my_fullname="$0"
myname=$(basename $0)
sec_day=86400
sec_week=604800
sec_year=31449600
sec_rotation=2419200
# sec_rotation is the default time length to save a tape.
# sec_rotation = (sec_day * 7) * 4
# As of writing, sec_rotation=2419200, or roughly 4-weeks.
# Let's begin
action="$1"
device="$2"
save_time="$3"
no_rw="${device}${no_rewind}"
if [ ! -e $no_rw ] ; then
   echo "Error: tape device $no_rw does not exist"
   exit 2
fi
# The following code segment is called only when the "init" module
# is executed.  The init module calls another copy of the program
# for each tape to initialize, but passes an "X" as the 3rd option
# indicating that we want a 1-second retention on the tape.
# In other words, by the time the initialization is completed,
# the tape is scratch.
rewind="mt -f $device rewind"
eject="mt -f $device rewoffl"
skip="mt -f $device fsf 1"
#tmp="/tmp/tape.$$.tmp"
$label_log "Label action $@"
if [ "$action" = "get" ] ; then
   $rewind
   /usr/bin/dd if=$device 2>/dev/null
fi
if [ "$action" = "make" ] ; then
   $rewind
   if [ "$save_time" = "X" ] ; then
      sec_rotation=1
   fi
   echo "Enter volser: \c"
   read volser
   s=$(${sec})
#   scratch=${sec_rotation}
   echo "$volser:$s:${sec_rotation}"|/usr/bin/dd of=$device 2>/dev/null
   $label_log "'make': $volser:$s:${sec_rotation}"
```

```
fi
if [ "$action" = "change" ] ; then
   if [ "$args" -eq "3" ] ; then
#      st=$(echo "$save_time*$sec_day"|bc 2>/dev/null)
      ((st = save_time * sec_day))
#      save_time=$st
      sec_rotation=$st
   fi
   $rewind
   current_label=$($my_fullname get $device)
   current_volser=$(echo "$current_label"|cut -f1 -d":")
   current_create=$(echo "$current_label"|cut -f2 -d":")
   current_scratch=$(echo "$current_label"|cut -f3 -d":")
   s=$($sec)
   scratch=$sec_rotation
   echo "$current_volser:$s:$scratch"|/usr/bin/dd of=$device 2>/dev/null
   $label_log "'change': $current_volser:$s:$scratch"
fi
if [ "$action" = "init" ] ; then
   clear
   file=$save_time
   if [ ! -r $file ] ; then
     echo "File $file does not exist.  Please check file and re-execute"
#      rm $tmp 1>/dev/null 2>&1
      exit 3
   fi
  echo "If you like to enter a description for these initialized tapes,"
   echo "enter it in the space below.  If you just press ENTER, no"
  echo "description will be entered.  This description will be used for"
   echo "all tapes in this initilization process."
   echo "For new tapes, enter a date initialized as a description."
   read Descr
   if [ -z "$Descr" ] ; then
      Descr="Reinitialized"
   else
      Descr1='echo "$Descr"|sed "s/ /_/g"'
      Descr="$Descr1"
   fi
   echo "Description will be: \"$Descr\" "
   for volser in 'cat $file|grep -v "#"' ; do
       $label_log "Reinit volser: $volser"
       echo "\nProcessing volser: $volser"
       echo "Writing new label"
       echo $volser |$my_fullname make $device X 1>/dev/null 2>&1
       echo "New label: `$my_fullname get $device`"
       echo "Updating tape database"
       $home/bin/db_update.ksh $device "$Descr"
       $eject
       if [ "$host" = "sapqam" ] ; then
           sleep 45
       else
           echo "Press ENTER after next tape is inserted"
           read myfoo
       fi
       echo " "
   done
   echo "Initialization completed"
fi
#rm $tmp 1>/dev/null 2>&1
```

TAPE_REPORT.KSH

```
#!/bin/ksh
# Menu for generating reports on the tape system
# Written 3/2/2001 Bill V.
home="/var/tapesys"
tapes="$home/db/tapes.db"
tape_history="$home/db/tape_history.db"
label_log="$home/bin/label_log.ksh"
sec="$home/bin/sec"
s2d="$home/bin/s2d"
format="%-13s%-29.26s%-29.26s%-10.10s%-5s%-40.38s%-5s\n"
z1="#"
hash1="#################################################################"
host='hostname'
if [ "$#" -ge "1" ] ; then
   batchf=1
   batch=$1
   report=$2
else
   batchf=0
fi
main() {
clear
echo $hash1
echo  "#                     Tape Library Processing                   #"
echo  "#                                                               #"
echo  "#     1.   Generate tape scratch report                         #"
echo  "#     2.   Generate list of all tapes                           #"
echo  "#     3.   Generate list of tapes for a specific backup         #"
echo  "#     4.   Generate list of tapes for a specific host           #"
echo  "#     5.   Generate tape history report                         #"
echo  "#     6.   Generate offsite tape report                         #"
echo  "#     7.   Generate onsite tape report                          #"
echo  "#     8.   Generate a daily tape movement report                #"
echo  "#     9.   Generate a report of all defective/discarded tapes   #"
echo  "#    10.   Generate a tape aging report                         #"
echo  "#    11.   Read a label on a tape currently in the tape drive   #"
echo  "#    12.   Mark a tape as \"BAD\" in the database              #"
echo  "#    13.   Mark a \"BAD\" tape as scratch in the database    #"
echo  "#    14.   Reset the scratch report pointer                    #"
echo  "#    15.   Create an advance scratch tape report               #"
echo  "#    20.   Backup the tape database                            #"
echo  "#    30.   Purge the old tape backups                          #"
echo  "#    50.   Mark tapes OFFSITE                                  #"
echo  "#    60.   Mark tapes ONSITE                                   #"
if [ "$ADM_USER" -eq "1" ] ; then
   echo  "# #####################################################     #"
   echo  "# #                 Administrative menu                #    #"
   echo  "# # 70.   Edit tape database                           #    #"
   echo  "# # 71.   Initialize tapes                             #    #"
   echo  "# # 73.   Generate list of tapes not offsite nor onsite#    #"
   echo  "# # 74.   Change retention time for a tape             #    #"
   echo  "# #                     Administrative menu            #    #"
   echo  "# #####################################################     #"
fi
echo  "#    99.   Exit                                                #"
echo  "#                                                              #"
echo $hash1
echo  "Please enter your selection (1-60,99): \c"
read menu
echo $menu
if [ "$menu" -eq "99" ] ; then
   exit 0
fi
if [ "$menu" -eq "3435" ] ; then
   if [ "$ADM_USER" -eq "0" ] ; then
      ADM_USER=1
   else
      ADM_USER=0
   fi
fi
if [ "$menu" -eq "1" -o "$menu" -eq "15" ] ; then
   scratch_report
fi
if [ "$menu" -eq "2" ] ; then
   all_tape_report
fi
if [ "$menu" -eq "3" ] ; then
   backup_report
fi
if [ "$menu" -eq "4" ] ; then
   host_report
fi
if [ "$menu" -eq "5" ] ; then
   history_report
fi
if [ "$menu" -eq "6" ] ; then
   offsite_report
```

```
fi
if [ "$menu" -eq "7" ] ; then
   onsite_report
fi
if [ "$menu" -eq "8" ] ; then
   daily_report
fi
if [ "$menu" -eq "9" ] ; then
   discard_report
fi
if [ "$menu" -eq "10" ] ; then
   aging_report
fi
if [ "$menu" -eq "11" ] ; then
   read_tape_label
fi
if [ "$menu" -eq "12" ] ; then
   make_tape_bad
fi
if [ "$menu" -eq "13" ] ; then
   make_tape_good
fi
if [ "$menu" -eq "14" ] ; then
   rm $home/reports/scratchreport.txt 2>/dev/null
fi
if [ "$menu" -eq "20" ] ; then
   backup_tape_db
fi
if [ "$menu" -eq "30" ] ; then
   purge_backup_tape_db
fi
if [ "$menu" -eq "50" ] ; then
   mark_offsite
fi
if [ "$menu" -eq "60" ] ; then
   mark_onsite
fi
if [ "$menu" -eq "70" -a "$ADM_USER" -eq "1" ] ; then
   manual_edit_db
fi
if [ "$menu" -eq "71" -a "$ADM_USER" -eq "1" ] ; then
   init_tapes
fi
if [ "$menu" -eq "73" -a "$ADM_USER" -eq "1" ] ; then
   other_tapes
fi
if [ "$menu" -eq "74" -a "$ADM_USER" -eq "1" ] ; then
   change_retention
fi
main
```

```
        }
discard_report() {
$label_log "Defective tape report"
clear
now=$(${sec})
now_date=$(${s2d} ${now})
tmp="/tmp/report.$$"
>$tmp
echo "Discarded/defective tapes as of: $now_date\n" >> $tmp
printf $format "Volser" "Date Defective" "Expiration time" "Host" "LOC"
"Contents" "   Usage" >> $tmp
printf $format "--------" "-------------" "-------------" "----" "--"
"--------" "   ----" >> $tmp
found=Ø
echo "Processing \c"
for tape_rec in 'cat $tapes|grep -i -E "discard|defect" |sort -n -k3,6'
; do
   volser='echo $tape_rec|cut -f1 -d":"'
   creation='echo $tape_rec|cut -f2 -d":"'
   lifetime='echo $tape_rec|cut -f3 -d":"'
   host='echo $tape_rec|cut -f4 -d":"'
   backup='echo $tape_rec|cut -f5 -d":"'
   location='echo $tape_rec|cut -f6 -d":"'
   creation_time='$s2d $creation'
   let foo=$creation+$lifetime
   expiration_time='$s2d $foo'
   passes='grep $volser $tape_history|wc -l'
   if [ "$location" -eq "Ø" ] ; then
      loc="ON"
   elif [ "$location" -eq "1" ] ; then
      loc="OFF"
   else
      loc="UNK"
   fi
   printf $format "$volser" "$creation_time" "$expiration_time" "$host"
"$loc" "$backup" "$passes" >> $tmp
   found='expr $found + 1'
   echo ".\c"
done
   clear
   echo "\nRecords found: $found\n" >>$tmp
   cat $tmp
   echo "\nWould you like a printed report "
   echo "If  Yes  then enter the printer name"
   echo "(or just press [ENTER] to NOT print the report)"
   read q
   if  [ -n "$q" ]  ; then
      $home/bin/printl.ksh $q $tmp
      sleep 3
   fi
   cp $tmp $home/reports/discardreport.txt
   rm $tmp
                 }
################################################################################
scratch_report() {
### log action
clear
### set now_date variable
now=$(${sec})
now_date=$(${s2d} ${now})
if [ "$menu" -eq "15" ] ; then
   echo "Enter number of days to post-date the scratch report"
   read scr_ndays
   if [ "${scr_ndays}X" != "X" ] ; then
      post_date='echo "$scr_ndays*86400"|bc'
      post_now='echo "$post_date+${now}"|bc'
      now=$post_now
      now_date='$s2d ${now}'
      rm $home/reports/scratchreport.txt 2>/dev/null
      $label_log "Post-dated scratch report for $now_date
(days=${scr_ndays})"
   fi
else
   $label_log "Scratch tape report"
fi
tmp="/tmp/report.$$"
>$tmp
echo "Scratch tapes as of: $now_date ($now)\n" >> $tmp
printf $format "Volser" "Creation time" "Expiration time" "Host" "LOC"
"Contents" "   Usage" >> $tmp
printf $format "--------" "-----------" "-------------" "----" "--" "-
-------" "   ----" >> $tmp
found=0
Gfound=0
lastvolser="X"
if [ ! -r $home/reports/scratchreport.txt ] ; then
   echo "Existing scratch report not found - creating a new one"
   most_recent_tape=2
   report_run=1
else
   most_recent_tape='cut -f2 -d":" $tapes|sort -n |tail -1'
   report_run='grep 'Scratch tapes' $home/reports/scratchreport.txt|cut
-f2 -d"("|cut -f1 -d")" 2>/dev/null'
fi
if [ "$most_recent_tape" -lt "$report_run" ] ; then
   rm $tmp
   cat $home/reports/scratchreport.txt
   echo "\nWould you like a printed report "
   echo "If  Yes  then enter the printer name"
   echo "(or just press [ENTER] to NOT print the report)"
   read q
   if  [ -n "$q" ]  ; then
      $home/bin/printl.ksh $q $home/reports/scratchreport.txt
      sleep 3
   fi
else
   echo "Processing \c"
   for tape_rec in $(cat $tapes|sort -n -k3,6) ; do

    volser=$(echo $tape_rec|cut -f1 -d":")
       creation=$(echo $tape_rec|cut -f2 -d":")
       lifetime=$(echo $tape_rec|cut -f3 -d":")
       ((foo = creation + lifetime))
      if [ "$lastvolser" = "X" ] ; then
         lastvolser=$(echo "$volser"|cut -c1-3)
      fi
         ### Tape is scratch
      if [ "$foo" -le "$now" ] ; then
           creation_time=$($s2d $creation)
           expiration_time=$(${s2d} ${foo})
           passes=$(grep -c $volser $tape_history)
         host=$(echo $tape_rec|cut -f4 -d":")
         backup=$(echo $tape_rec|cut -f5 -d":")
         location=$(echo $tape_rec|cut -f6 -d":")
         if [ "$location" -eq "Ø" ] ; then
               loc="ON"
         elif [ "$location" -eq "1" ] ; then
               loc="OFF"
```

Editor's note: this article will be concluded next month.

Bill Verzal
Project Leader
Komatsu America (USA)

Why not share your expertise and earn money at the same time? *AIX Update* is looking for shell scripts, program code, JavaScript, etc that experienced users of AIX have written to make their life, or the lives of other users, easier. Articles can be of any length and should be e-mailed to the editor, Trevor Eddolls, at trevore@xephon.com.

AIX news

Dassault Systemes and IBM have announced Version 5 Release 15 of their Product Life-cycle Management (PLM) portfolio, comprising CATIA for collaborative product development, and ENOVIA and SMARTEAM for product data and life-cycle management, collaboration, and decision support.

Concurrently, Dassault Systemes announced V5R15 of DELMIA for digital development of factory and production processes.

The new releases provide unified working environments or desktops, targeting the specific needs of user communities such as engineering, manufacturing, and enterprise users.

V5R15 uses the open and lightweight 3DXML format, which means that customers can communicate in 3D by sharing 3D representations of the virtual product throughout the extended enterprise. V5R15 also introduces a new 3D XML viewer.

For further information contact:
URL: www.3ds.com/V5R15, www.ibm.com/software/plm.

* * *

Cendura has announced that its Dependency Visualization module, which captures and maps the dependencies and inter-relationships between IT assets, is now available as a part of the Cohesion Suite. The new module provides IT organizations with comprehensive dependency and relationship visualization across complex, distributed, multi-tiered, and custom-developed application infrastructures.

The module supports the IBM Tivoli Discovery Library, part of IBM's new IT Service Management solutions for helping enterprises better align hardware and software resources with business needs.

Cohesion allows IT organizations to discover, track, and take inventory of the tens of thousands of configuration items existing in complex enterprise applications. With this information users can quickly troubleshoot configuration-related problems, proactively institute IT controls over those configurations to harden their infrastructure, and enforce best practices for application management.

Applications supported include AIX 5L, DB2 UDB, Informix Database Server, IBM HTTP Server, WebSphere MQ, and WAS.

For further information contact:
URL: www.cendura.com/news/030805.html.

* * *

IBM has announced Version 3.2 of Parallel Engineering and Scientific Subroutine Library (ESSL) for AIX 5L, its scalable mathematical subroutine library.

The product is tuned for optimum performance on clusters of POWER4 and POWER5 servers connected with the IBM pSeries High Performance Switch, and tuned for BladeCenter JS20 processors and POWER5 servers connected with a Myrinet-2000 Switch with Myrinet/PCI-X adapters.

For further information contact:
URL: www.ibm.com/servers/eserver/clusters/software/order_guide.pdf.

* * *