# 102

AIX update

*April 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.



## 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.*

# Excluding a boot disk from SDD

SDD is the Subsystem Device Driver software, supplied by IBM, which allows vpath devices to be created for one or more hdisks, in an ESS storage environment. For example, if you have two fibre attachments to your SAN or ESS, two hdisks will be created for the same physical disk. SDD creates a device, called vpath, which it uses to balance activity across the two adapters and to the physical device. It also serves to allow path redundancy, should one path not be available.

One of the catches is that when you boot your system, SDD is not running, and therefore vpath devices cannot be used as a rootvg disk. On some of our smaller servers we wanted to be able to mirror rootvg to something other than the internal drives, if the internal drives are on one SCSI adapter.

The way to accomplish this in an SDD/ESS environment is to use the **querysn** command to exclude the disk in question from being controlled by SDD.

The steps to accomplish this are as follows:

- Assign the LUN in question to only on host adapter (and accompanying SAN zoning if needed).
- Run **cfgmgr**.
- Note the hdisk and vpath created.
- **rmdev –dl vpathx** (vpathx is the vpath created in the previous step).
- **querysn –l hdiskn** (hdiskn is the hdisk created in the previous step).
- Verify that the file */etc/vpexclude* was created.
- Add the hdisk to rootvg, mirror to it, etc.

Command output examples:

```
# querysn -l hdisk4
hdisk4 SERIAL_NUMBER = 11E25492
# cat /etc/vpexclude
hdisk4 SERIAL NUMBER = 11E25492
```

*David Miller*
*Database Architect*
*Baystate Health Systems (USA)*



# Using HMC command line interface to manage Logical Partitioning Facility (LPAR) – part 2

*This month we conclude the article looking at HMC commands.*

## lshwinfo

The **lshwinfo** command will display hardware information such as temperature of the managed system:

```
lshwinfo -r sys -e frame-name -n object-name [ | –all ] [-F
<format> ] [ –help]
```

where:

- **-r** – the resource type to display. A valid value is *sys* for system.
- **-e** – the name of the frame the system is in.
- **-n** – the name of the object to perform the listing on. This parameter cannot be specified with **-all**.
- **-all** – list all the objects of a particular resource type. This parameter cannot be used with **-n**.
- **-F** – if specified, has a delimiter-separated list of property names to be queried. Valid values are *temperature*, *current*, *voltage*, *power*, and *total_power*.
- **-help** – prints a help message.

Example command:

```
[hmcusr@hmcproj hmcusr]$ lssyscfg -r frame –all
Name               FrameNum  IsReal  PortNums  Frame Type
F7Ø38-6M2*C25699A  Ø         No                4
[hmcusr@hmcproj hmcusr]$ lshwinfo -r sys -e F7Ø38-6M2*C25699A –all
Either the connection to the bulk power assembly was lost, or there is
no bulk  power assembly. Retry the operation.
This command is not valid for our system.
```

## lshwres

The **lshwres** command will list hardware resources configuration:

```
lshwres -m "managed-system" [-p "partition-name" | –all ]
-r [ resource-type ] [ -y "led-type" ] [ -F <format>  ]  [–help ]
```

where:

- **-m** – the name of the managed system, where the hardware resource is configured. If there are multiple managed systems with the same user-defined name, specify the managed system name enclosed in double quotes and in the form mmmm-ttt*sssssss, where *mmmm* is the machine type, *ttt* is the model type, and *sssssss* the serial number of the managed system.
- **-p** – the user-defined name of the partition where the hardware resource will be queried.
- **-all** – retrieve information from all partitions on the given managed system.
- **-r** – the hardware resource type to view. Possible values are *ALL*, *cpu* (processor), *mem* (memory), *slot* (PCI slot), or led.
- **-y** – type of led to list. To list system attention led, use the keyword *sys*. To list identify led, use the keyword *identify*.
- **-F** – if specified, produces a delimiter-separated list of names representing the desired properties to be queried. Valid values are *system*, *name*, *key*, *state*, *status*, *id*, *parent*, *location*, *classcode*, assigned_to, index, location_code, max, min, allocated,free, *lmb_size*, *drawer_id*, *slot_id*, *slot_type*, *phys_loc. partition*, and *partition_name*.

- **-help** – prints a help message.

Some examples of usage of this command are shown below.

List of all processors on a managed system:

```
[hmcusr@hmcproj hmcusr]$ lshwres -m proj -r cpu
id  Status                partition             assigned_to
25  Configured by System  Ø01*7Ø38-6M2*C25699A  proja
24  Configured by System  Ø02*7Ø38-6M2*C25699A  projb
17  Configured by User    Ø01*7Ø38-6M2*C25699A  proja
16  Configured by System  Ø02*7Ø38-6M2*C25699A  projb
9   Configured by System  Ø02*7Ø38-6M2*C25699A  projb
8   Configured by System  Ø02*7Ø38-6M2*C25699A  projb
1   Configured by System  Ø02*7Ø38-6M2*C25699A  projb
Ø   Configured by System  Ø02*7Ø38-6M2*C25699A  projb
```

List the memory resources of partition proja:

```
[hmcusr@hmcproj hmcusr]$ lshwres -m proj -r mem -p proja
allocated  free  lmb_size  max  min  partition           system
partition_name

16         Ø     256       32   8    Ø01*7Ø38-6M2*C25699A  proj  proja
Check system attention LED status:
[hmcusr@hmcproj hmcusr]$ lshwres -m proj -r led -y sys
index   State  location_code
884738  off    UØ.1
```

## chhwres

The **chhwres** command will change the hardware resource configuration. It allows dynamic reconfiguration of processor, memory, and I/O slots:

```
       chhwres -r [ mem | cpu | slot | led ] -o [ a | r | m | s ]
       -m "managed-system" [ -p "source-partition-name"  ]
[  -t "target-partition-name"  ]  [ -i "drawer-id" ]
       [ -l physical-location-code ] [ -s <slot-id> ] [ -q <quantity>  ]
       [-w  <timeout> ] [ -v <LED setting> ] [ -d <detail-level> ]
       [ -x <LED index> ] [ -y <LED type> ] [ –help ]
```

where:

- **-r** – the hardware resource type to change. Specify *mem* for memory, *cpu* for processor, *slot* for I/O slot, and *led* for led.

- **-o** – the operation to perform. For adding a hardware resource specify *a*; for removing a hardware resource, *r*; for moving resources, *m*; and for setting led values, *s*.
- **-m** – the name of the managed system where the hardware resource is configured. This name must be enclosed in double quotes and be in the form mmmm*sssss, where *mmmm* is the machine model type and *sssss* is the machine's serial number.
- **-p** – the user-defined name of the partition to perform the operation.
- **-t** – the user-defined name of the partition to which to move the new hardware resource.
- **-v** – the value to which to set the led.
- **-i** – the drawer ID. This argument needs to be specified for slot resource only.
- **-l** – the physical location code of the I/O slot. This argument cannot be specified with the **-i** and **-s** flags.
- **-s** – the slot ID. This argument needs to be specified for a slot resource only.
- **-q** – the quantity of hardware resource to change. For processor, this value will specify the number of processors to add, remove, or move. For memory, this value will specify the number of LMBs.
- **-w** – the timeout value to be used by the Dynamic Resource Manager command running on the partition.
- **-s** – the slot ID. This argument needs to be specified for a slot resource only.
- **-q** – the quantity of hardware resource to change. For processor, this value will specify the number of processors to add, remove, or move. For memory, this value will specify the number of LMBs.

- **-w** – the timeout value to be used by the Dynamic Resource Manager command running on the partition. The default value is 0, indicating that no timeout is used, and that the command on the partition will take as much time as it needs to complete the operation.
- **-x** – the index number of the led to set.
- **-y** – the led type. For system attention, specify *sys*; for identify, specify *ident*.
- **-d** – the detail level to be used by the Dynamic Resource Manager command (**drmgr**) running on the partition. Valid values are 0 to 5. The default value is 1.

The following command removes three processors from partition p1:

```
chhwres -m "7040-681*8386522" -p "p1" -r cpu -o r -q 3
```

The following command moves 1GB of RAM (4 LMBs) from partition p1 to partition p2:

```
chhwres -m "7040-681*8386522" -p "p1" -t "p2"  -r  mem -o m -q 4
```

The following command removes one I/O slot from partition p1:

```
chhwres  -m "7040-681*8386522" -p "p1" -r slot -o r –l U1.9-P13-I7
```

### mkvterm

The **mkvterm** command will open a virtual terminal session to a partition or service processor (in case the managed system is powered off). After establishing a virtual terminal session, the command **rmvterm** can be used to close the session, or the '~.' characters sequence can be entered at the current session to terminate it:

```
mkvterm  -m  "managed-system"  [  -p  "partition-name" ] [ –help ]
```

where:

- **-m** – the name of the managed system to open a virtual terminal session on. If there are multiple managed systems

with the same user-defined name, specify the managed system name enclosed in double quotes and in the form mmmm-ttt*sssssss, where *mmmm* is the machine type, *ttt* is the model type, and *sssssss* the serial number of the managed system.

- **-p** – the user-defined name of the partition on which to open a virtual terminal session.
- **-help** – prints a help message.

The following command opens a virtual terminal session on partition proja:

```
[hmcusr@hmcproj hmcusr]$ mkvterm -m proj -p proja
NVTS hmcproj 9734 001*7038-6M2*C25699A 1 001*7038-6M2*C25699A _VT_
AIX Version 5
(C) Copyrights by IBM and by others 1982, 2002.
Console login:
```

### rmvterm

The **rmvterm** command will remove (close) a virtual terminal session:

```
rmvterm -m "managed-system" [ -p "partition-name" ] [–help ]
```

where:

- **-m** – the name of the managed system for which the virtual terminal session is to be closed. If there are multiple managed systems with the same user-defined name, specify the managed system name enclosed in double quotes and in the form mmm*ssss, where *mmm* is the machine type and *ssss* the serial number of the managed system.
- **-p** – the user-defined name of the partition for which the virtual terminal session is to be closed.
- **-help** – prints a help message.

The following command closes a virtual terminal session on partition proja:

```
rmvterm -m proj -p proja
```

## rmsplock

The **rmsplock** command will remove a lock set in the service processor in the redundant HMC configuration. In a very rare situation, HMC might fail to remove the lock set in the service processor on a managed system. This command removes all leftover locks:

```
rmsplock -m "managed-system" [ –help ]
```

where:

- **-m** – the name of the managed system on which to remove the lock. If there are multiple managed systems with the same user-defined name, specify the managed system name enclosed in double quotes and in the form mmm*ssss, where *mmm* is the machine type and *ssss* the serial number of the managed system.
- **-help** – prints a help message.

The following command removes any lock set in the service processor:

```
rmsplock -m proj
```

## rsthwres

The **rsthwres** command will restore the hardware resource configuration following a failure in a dynamic LPAR reconfiguration. When a DLPAR operation fails to complete successfully, the hardware resource assignment in NVRAM of the managed system and in the AIX OS running in the partition become inconsistent, and this command fixes the condition:

```
rsthwres  -m "managed-system" [ -p "partition-name" ]
                                                -r [cpu | mem | slot ]
  [ -u <processor id> ]
  [[ -i <drawer id> -s <slot id> ] | -l "physical location code" ]
                                                              [ –help ]
```

where:

- **-m** – the name of the managed system where the hardware

resource is configured. This name must be enclosed in double quotes and in the form mmm*ssss, where *mmm* is the machine type and *ssss* the serial number of the managed system.

- **-p** – the user-defined name of the partition on which to perform the restore. This argument must be enclosed in double quotes.
- **-r** – the hardware resource type to restore. Possible values are *cpu* (processor), *mem* (memory), or *slot*.
- **-u** – the processor ID of the processor to restore.
- **-i** – the machine type and serial number of an I/O slot's scoping drawer. This argument should not be specified if the resource type is not *slot*.
- **-s** – the slot number of a PCI slot to restore.
- **-l** – the physical location code of the I/O slot. This argument cannot be specified with the **-i** and **-s** flags.
- **-help** – prints a help message.

The following command restores memory on partition p1:

```
rsthwres -m proj -r mem -p proja
```

```
[hmcusr@hmcproj hmcusr]$ rsthwres -m proj -r mem -p proja
The partition entered does not require any memory to be restored.
```

The above message is produced if no actual inconsistency has been detected.

### chsysstate

The **chsysstate** command is used to change the system state. By changing the system state it is possible to perform various operations on managed system and partitions such as: power on, power off, reset, activate, shutdown, etc.

```
chsysstate  -r
```

```
[sys|lpar|sysprof] -o on |off|reset|rebuild|osshutdown|recover
-m  "managed-system"  [ -n "object-name" ] [ -f profile-name ]
[ -c full | lpar ] [-b norm | dd | sms | of | ds | std | auto ] [–help ]
```

where:

- **-r** – the system resource type to modify. Valid values are *lpar* for partition, *sys* for managed system, and *sysprof* for system profile.
- **-m** – the name of the managed system where the resource is configured. If there are multiple managed systems with the same user-defined name, specify the managed system name enclosed in double quotes and in the form mmmm-ttt*sssssss, where *mmmm* is the machine type, *ttt* is the model type, and *sssssss* the serial number of the managed system.
- **-n** – the name of the system resource object, whose state will be modified.
- **-o** – the operation to be performed on the object. Valid values are:
  - **on** – to power on the managed system or to activate a partition.
  - **off** – to power off a managed system or to hard reset a partition.
  - **reset** – to perform a soft reset on a partition.
  - **rebuild** – to rebuild a managed system.
  - **osshutdown** – to shut down the operating system on the partition.
  - **recover** – to recover partition data.
- **-b** – the boot setting option to use when powering on the managed system. Valid values are:
  - **norm** – normal.

  - **dd** – diagnostic default boot list.
  - **sms** – SMS.
  - **of** – OpenFirmware OK prompt.
  - **ds** – diagnostic stored boot list.
  - **std** – partition standby.
  - **auto** – automatically start partitions.
- **-c** – the mode to power on the managed system. Valid values are:
  - **full** – full system partition.
  - **lpar** – logical partition or to hard reset a partition.
- **-f** – the name of the profile to use when activating a partition.
- **-help** – prints a help message.

The following command activates logical partition proj, using partition profile proj_normal:

```
chsysstate -m proj -r lpar -o on -n ep65Ø2a -f proja_normal
```

The following command powers on the managed system proj in partition proja:

```
chsysstate -n proj -r sys -o on -c proja -b std
```

The following command activates a system profile:

```
chsysstate -r sysprof -m proj -o on -n sysprof1
```

## lssvcevents

The **lssvcevents** command will display the hardware serviceable events or HMC console events:

```
lssvcevents -t { hardware | console } [ -d <number-of-days-to-go-back> ]
       [ -m "managed-system" -s {ALL  |  sp  | lpar}
       [ -p "partition-name" ]] [ -w {ALL | "switch-mtms" } ]
                                              [ -F <format> ] [ –help ]
```

where:

- **-t** – the type of events to display. Valid values are *hardware* for serviceable events, or *console* for console type events.
- **-d** – the number of days to go back to search for serviceable events. By default, 7 will be used.
- **-m** – the name of the managed system to display events from. If there are multiple managed systems with the same user-defined name, specify the managed system name enclosed in double quotes and in the form mmm*ssss, where *mmm* is the machine type and *ssss* is the serial number of the managed system. This parameter is valid only for hardware (serviceable) events and cannot be specified with the **-w** parameter.
- **-s** – the source of the events to display. Valid values are:
  - **sp** – to display events from the service processor.
  - **lpar** – to display events from the partitions.
  - **ALL** – to display all events.

  This parameter is valid only with the **-m** parameter.
- **-p** – the name of the partition that the serviceable events come from. This parameter is valid only with the **-s** parameter.
- **-w** – the switch to display events from. Specify *ALL* to view events from all switches, or specify a single switch enclosed in double quotes and in the form mmmm*ssss, where *mmmm* is the machine type, and *ssss* is the serial number of the switch. This parameter is valid only for hardware (serviceable) events and cannot be specified with the **-m** parameter.
- **-F** – if specified, use a delimiter-separated list of property names to be queried. Valid values are *name*, *time*, *callhome*, *calledhome*, *errorclass*, *description*. When this option is specified, the output will be a delimiter-separated list of the values requested.

- **-help** – prints a help message.

The following command will display events from the console:

```
 [hmcusr@hmcproj hmcusr]$ lssvcevents -t console
Earliest Timestamp    Description
12/11/03 06:36:29 AM  HSCE2014 UserName hscroot Virtual terminal has
been open  on partition proj of lpar id 7038-6M2*C25333A of managed
system proj;
12/11/03 06:35:53 AM  HSCE2174 User hscroot login was successful from
remote host hmc65001.mycompany.com with IP address 199.221.37.55.

12/06/03 11:00:27 AM  HSCE2016 UserName hscroot Logical Partition projb
with ID 002 of managed system proj has been activated with profile prof;
12/06/03 11:00:26 AM  HSCE2163 User hscroot: Completed activating
partition.
12/06/03 11:00:05 AM  HSCE2026 11264 of memory from managed system proj
with ID of 7038-6M2*C25699A has been assigned to LPAR projb with ID 2;
```

## REFERENCES


1 AIX 5L Version 5.2 *AIX Installation in a Partitioned Environment*, SC23-4382. IBM Corporation.

2 *Effective System Management Using the IBM Hardware Management Console for pSeries*, SC24-7038, IBM Corporation.

3 *Planning for Partitioned-System Operations*, SA38-0626, IBM Corporation.

4 *Site and Hardware Planning Information*, SA38-0508, IBM Corporation.

5 *IBM Hardware Management Console for pSeries Installation and Operations Guide*, SA38-0590, IBM Corporation.

6 *Electronic Service Agent for eServer pSeries User's Guide*, LCD4-1060, IBM Corporation.

7 *PCI Adapter Placement* reference, SA38-0538, IBM Corporation.



*Alex Polak*
*System Engineer*
*APS (Israel)*

# Command line processing in shell script

## INTRODUCTION

Command line processing in shell script is often not robust. Also, the way the parameters are provided to the shell script is widely varied. Quite often, we're asked to run a script that does not offer a full explanation of what parameters it takes and how to provide these parameters. In some cases, one has to delve into the script in order to establish the details of any parameters, which may have some validations attached, which in turn may not have been fully explained.

When a script takes a number of parameters, some of which may be mandatory and some may be optional, the best way to provide these parameters is as (name=value) pairs as follows:

```
<script_name>     arg1=<value>   arg2=<value>   arg3=<value>
```

These parameters should include a parameter called *help*, for which allowed values are Y or N . In order to see all the parameters and their allowable values the user should be able to execute:

```
<script_namne>  help=y
```

This name=value pair method, of providing parameters is clearly understood by the user, who can execute the script without knowing much about it.

Any script writing to log and/error files should name these files after the script and with the proper file extensions (ie log and err). These require the log and error file names to be defined using variable references (see the listing below) and then re-evaluate them afterwards.

The article contains a listing of a script pcl.ksh, which demonstrates the name=value pair method of providing parameters to a script. The script can be incorporated into any other scripts by amending the variable ${VALID_ARGUMENT_LIST}, which contains all

allowable arguments, validation indicators, validation types, and any possible values.

The script makes extensive use of the idea of variable referencing, which is a very powerful way of using shell variables.

## USE OF VARIABLE REFERENCES

Let's start with an example.

Whenever we display a message, we want to display the current date and time and not a previously stored date and time.

Let's define two message prefixes as follows:

```
TIME_NOW='date  '%d/%m/%Y  %H:%M:%S'
ERROR="script:ERROR:${TIME_NOW}:"
INFO="scriptame:INFO:${TIME_NOW}:"
```

Let's display a message:

```
echo   "${ERROR} Invalid   directory"
```

This message will be displayed with a timestamp that was captured at a time the variable TIME_NOW was assigned, but we want current date and time (ie at the time of the display).

Let's display the message as follows:

```
TIME_NOW='date  '%d/%m/%Y  %H:%M:%S'
echo   "${ERROR} Invalid   directory"
```

This message will still be displayed with a timestamp that was captured at the time the variable $TIME_NOW was assigned because although we've reassigned the variable, $TIME_NOW, the variable $ERROR contains the old value of variable $TIME_NOW.

To overcome this problem, we need to do the following:

```
ERROR="script:ERROR:\${TIME_NOW}:"
INFO="scriptame:INFO:\${TIME_NOW

TIME_NOW='date  '%d/%m/%Y  %H:%M:%S'
echo   "eval    echo  ${ERROR} Invalid   directory"
```

The keyword is the command **eval** (for evaluate). Every time the command **eval echo ${ERROR}** is issued, it will contain the current date and time because $TIME_NOW will be assigned the current date and time and the **eval** command will re-evaluate variable $L_TIME_NOW to establish its current value.

## SUMMARY OF VARIABLE REFERENCING USAGE

How to define a variable that will contain another variable:

```
VAR2="var2"
VAR1=\${VAR2}
```

How to assign the value of a variable that is contained in another variable:

```
L_VAR2='eval  echo ${VAR1}'   #    value  should  be   var2
```

This is also known as dereferencing a variable. This can be nested as deep as required.

How to define a variable that will contain a second variable, which will contain another variable:

```
VAR3="var3"
VAR2=\${VAR3}
VAR1=\${VAR2}
```

How to assign a value of a variable that is contained in another variable, which in turn is contained in another variable:

```
L_VAR3='eval  eval   echo ${VAR1}'   #  value should be var3
```

How to assign a value to a variable that is contained in another variable:

```
VAR2=""
VAR1=VAR2          #  variable VAR2 must be assigned without the $ sign
eval   $VAR1 ="var2"
echo $VAR2         #  value should be var2
```

How to assign a value to a variable that is contained in another variable, which in turn is contained in another variable:

```
VAR3=""
VAR2=VAR3
```

```
VAR1=VAR2
eval   eval   $VAR1 ="var3"

echo $VAR3
#value should be var3 but you will notice the value is null and variable
#  $VAR2 contains the value var3. This is because for the
#  assignment purpose, the recursive dereferencing does not go beyond
#  one recursion.
```

## LISTING OF PARSECOMMANDLINE.KSH

```
#########################################################################
#  Name       :  pcl.sh ( Parse command line )
#
#  Overview   :  The script demonstrates the name=value pair method
#                of providing parameters to a shell script .
#
#  Notes      :  1. The script contains the following functions:
#                                        o  main
#                                        o  ParseCommandLine
#                                        o  InitialiseVariables
#                                        o  ValidateArgument
#                                        o  StoreArgumentValue
#                                        o  ReEvaluateVariables
#                                        o InitialiseLogAndErrorFiles
#                                        o  ShowParameters
#                                        o  ProcessExit
#
#
#               2. The  script defines the following variables using
#                        the idea of variable referencing as follows:
#
#        o TIME_NOW="\$(date '+%d/%m/%Y %H:%M:%S')"
#        o ERROR="\$(basename ${SCRIPT_NAME}):ERROR:${TIME_NOW}:"
#        o INFO="\$(basename  ${SCRIPT_NAME}):INFO:${TIME_NOW}:"
#        o TEMP_FILE="\$(basename \${SCRIPT_NAME} | sed \
#                                              s/\\\..*$//).tmp"
#        o SCRIPT_LOG_FILE="\$(basename \${SCRIPT_NAME} | sed  \
#                                              s/\\\..*$//).log"
#        o SCRIPT_ERROR_FILE="\$(basename \${SCRIPT_NAME} | sed \
#                                              s/\\\..*$//).err"
#        o SCRIPT_LOG_DIR="\$(dirname \${SCRIPT_NAME})"
#        o SCRIPT_ERROR_DIR="\$(dirname \${SCRIPT_NAME})"
#########################################################################

#########################################################################
#    Name      :  InitialiseVariables
#    Overview  :   The function initializes all the required
```

```
#                  variables
#    Notes     :
#############################################################################
InitialiseVariables ()
{
#
#   function return codes
#
TRUE=Ø
FALSE=1
#
#  exit  code
#
SEC=Ø    #  success
FEC=1    #  failure
#
# terminal capabilities
#
BOLDON='tput smso'  ; export BOLDON
BOLDOFF='tput rmso' ; export BOLDOFF
ESC="\ØØ33["
#
TIME_NOW="\$(date '+%d/%m/%Y %H:%M:%S')"
#
ERROR="\$(basename ${SCRIPT_NAME}):ERROR:${TIME_NOW}:"
INFO="\$(basename  ${SCRIPT_NAME}):INFO:${TIME_NOW}:"
#
# store allowed arguments in the  list
#
#<argument name>:<param name for argument value>:<validation req
ind>:<validation type:<list of values>:
#
# validation type
#
#    F=file existence validation (file with full path name)
#    D=directory existence validation
#  LOV=list of values, shown in the next column with pipe separators
#
#
VALID_ARGUMENT_LIST="prog:P_PROG:Y:F:\${P_SCRIPT_DIR}/\${P_PROG}:   \
              prog_dir:P_SCRIPT_DIR:Y:D::                           \
              log_file:P_LOG_FILE:Y:F::                             \
              log_dir:P_LOG_DIR:Y:D:                                \
              err_file:P_ERR_FILE::::                               \
              err_dir:P_ERR_DIR:Y:D::                               \
              sqltrace:P_SQLTRACE:Y:LOV:Y|N:                        \
              verbose:P_VERBOSE:Y:LOV:Y|N:                          \
              ocf:P_OCF::::                                         \
              ocf_dir:P_OCF_DIR:Y:D::                               \
```

```
                termout:P_TERMOUT:Y:LOV:Y|N:                          \
                spdebug:P_SPDEBUG:Y:LOV:Y|N:                          \
                abdebug:P_SPDEBUG:Y:LOV:Y|N:                          \
                utrace:P_SPDEBUG:Y:LOV:Y|N:                           \
                abdebug:P_SPDEBUG:Y:LOV:Y|N:                          \
                loop:P_SPDEBUG:Y:LOV:Y|N:                             \
                restart:P_RESTART:Y:LOV:Y|N:                          \
                help:P_HELP:Y:LOV:Y|N:                                "
#
ARG=""   #  argument name from command line
VAL=""   #  argument value from command line
#
TEMP_FILE="\$(basename \${SCRIPT_NAME} | sed s/\\\..*$//).tmp"
#
#  default log and error files, using $SCRIPT_NAME
#
SCRIPT_LOG_FILE="\$(basename \${SCRIPT_NAME} | sed s/\\\..*$//).log"
SCRIPT_ERROR_FILE="\$(basename \${SCRIPT_NAME} | sed s/\\\..*$//).err"
#
# default log and error directories using $SCRIPT_NAME
#
SCRIPT_LOG_DIR="\$(dirname \${SCRIPT_NAME})"
SCRIPT_ERROR_DIR="\$(dirname \${SCRIPT_NAME})"
#
# messages
#
ARG_VALUE_MISSING="Must provide a value for argument \${ARG_NAME} as
\${ARG_NAME}\=\<value\>"
INVALID_ARGUMENT="\${ARG_NAME} is an invalid argument"
INCORRECT_ARG_SPECIFICATION="Argument must be specified as
\<argument\>\=\<value\>"
ARG_NAME_MISSING="Argument name missing"
DUP_ARG_SPECIFIED="Duplicate argument \${ARG_NAME}\=\${ARG_VAL}
specified"
DIR_NOT_EXISTS="Directory , \${ARG_VAL} does not exist for parameter
\${ARG_NAME}"
OS_ERROR="\${ERR_MSG}"
}
#########################################################################
#    Name     : MoveCursor
#    Input    : Y and X coordinates
#    Returns  : None
#    Overview : It moves the cursor to the required location (Y,X).
#    Notes    :
#########################################################################
MoveCursor  ( )
{
YCOR=$1
XCOR=$2
```

```
echo     "${ESC}${YCOR};${XCOR}H"
}
#########################################################################
#  Name     : DisplayMessage
#  Overview : The function displays the message
#  Input    : 1. Message type (E = Error, I = Informative)
#             2. Error Code as defined in DefineMessages ().
#             3. Message to be acknowledged flag  (Y=yes N=no)
#  Notes    :
#########################################################################
DisplayMessage ( )
{
MESSAGE_TYPE=$1
MESSAGE_TEXT='eval echo $2'
ACKNOWLEDGE_FLAG="$3"
#
# re-evaluate message prefixes
#
L_ERROR='eval echo ${ERROR}'
L_INFO='eval echo  ${INFO}'
#
# default the message acknowledge flag
#
if [ "${ACKNOWLEDGE_FLAG}" = ""  ]
then
  ACKNOWLEDGE_FLAG="N"
fi
#
clear
MoveCursor 2Ø 1
if [ "${MESSAGE_TYPE}" = "E" ]
then
  if [ "${ACKNOWLEDGE_FLAG}" = "Y"  ]
  then
     echo  "${BOLDON}${L_ERROR}${MESSAGE_TEXT}${BOLDOFF}\c"
  else
     echo  "${BOLDON}${L_ERROR}${MESSAGE_TEXT}...${BOLDOFF}\c"
  fi
else
  if [ "${ACKNOWLEDGE_FLAG}" = "Y"  ]
  then
     echo  "${BOLDON}${L_INFO}${MESSAGE_TEXT}${BOLDOFF}\c"
  else
     echo  "${BOLDON}${L_INFO}${MESSAGE_TEXT}...${BOLDOFF}\c"
  fi
fi
#
# examine message acknowledge flag
#
```

```
if [ "${ACKNOWLEDGE_FLAG}"  =  "Y"  ]
then
      read  DUMMY
fi
return ${TRUE}
}
#########################################################################
#  Name     : StoreArgumentValue
#  Overview : The function stores the value for a specific argument
#  Input    : 1. argument name
#             2. argument value
#  Returns  : $TRUE  if argument is not a duplicate
#             $FALSE if argument is a duplicate
#  Notes    :
#########################################################################
StoreArgumentValue ()
{
#
# assign parameters
# convert argument to upper case
#
P_ARGUMENT='echo $1 | tr ["a-z"] ["A-Z"]'
P_VALUE="$2"
#
# check against the list
#
for VAR in ${VALID_ARGUMENT_LIST}
do
   #
   ARGUMENT='echo "${VAR}" | cut -d':' -f1 | tr ["a-z"] ["A-Z"]'
   if [ "${P_ARGUMENT}" = "${ARGUMENT}" ]
   then
      #
      # argument found
      #
      # get the variable that will store the argument's value
      # and assign it to a pointer variable
      #
      P_VALUE_VAR='echo "${VAR}" | cut -d':' -f2'
      if [  "`eval  echo \\$${P_VALUE_VAR}`"  = "" ]
      then
           #
           # pointer variable points to value variable for this argument
           # that is currently null and therefore stores its value
           #
           eval ${P_VALUE_VAR}=${P_VALUE}
           #
           if [ "${P_ARGUMENT}"  = "HELP"  ]
           then
```

```
                ShowParameters
            fi
            return $TRUE
        else
           #
           # value already stored; duplicate argument specified
           #
           DisplayMessage E "${DUP_ARG_SPECIFIED}"
           return $FALSE
        fi
     fi
done
}
#############################################################################
#  Name     : ValidateArgument
#  Overview : The function validates the argument name and its value
#  Input    : 1. argument name
#  Returns  : $TRUE  if argument is valid
#             $FALSE if argument is invalid (not in
#             $VALID_ARGUMENT_LIST )
#  Notes    :
#############################################################################
ValidateArgument ()
{
#
# assign parameter and convert it to upper case
#
P_ARG_NAME='echo $1 | tr ["a-z"] ["A-Z"]'
P_ARG_NAME_UPPERCASE='echo $1 | tr ["a-z"] ["A-Z"]'
#
P_ARG_VAL_UPPERCASE='echo $2 | tr ["a-z"] ["A-Z"]'
P_ARG_VAL="$2"
#
if [ "${P_ARG_NAME}" =  ""   ]
then
     #
     # no argument name before = sign
     #
     DisplayMessage E "${ARG_NAME_MISSING}"
     return $FALSE
fi
#
if [ "${P_ARG_VAL}" =  ""   ]
then
     #
     # no value provided for this argument
     #
     DisplayMessage E "${INCORRECT_ARG_SPECIFICATION}"
     return $FALSE
```

```
fi
#
# check against the list, $VALID_ARGUMENT_LIST
#
VALID_ARGUMENT="N"
for VAR in ${VALID_ARGUMENT_LIST}
do
   #
   L_ARG_NAME='echo "${VAR}" | cut -d':' -f1 | tr ["a-z"] ["A-Z"]'
   ARG_PARAM_NAME='echo "${VAR}" | cut -d':' -f2 | tr ["a-z"] ["A-Z"]'
   VAL_REQ_IND='echo "${VAR}" | cut -d':' -f3 | tr ["a-z"] ["A-Z"]'
   VAL_TYPE='echo "${VAR}" | cut -d':' -f4 | tr ["a-z"] ["A-Z"]'
   VAL_VALUE='echo "${VAR}" | cut -d':' -f5 | tr ["a-z"] ["A-Z"]'
   #
   #
   if [ "${P_ARG_NAME_UPPERCASE}" = "${L_ARG_NAME}" ]
   then
       VALID_ARGUMENT="Y"
       break
    fi
done
#
if [ "${VALID_ARGUMENT}" = "N" ]
then
    # invalid argument
    #
    DisplayMessage E "${INVALID_ARGUMENT}"
    return $FALSE
fi
#
# validate for allowed values
#
if [  "${VAL_REQ_IND}"  = "Y"  ]
then
    #
    # validation required for this argument
    #
    if [ "${VAL_TYPE}"    = "LOV"  ]
    then
        #
        # validate against allowed list of values
        #
        FIELD_NO=1
        NEXT_LOV_VAL="X"
        while [ "${NEXT_LOV_VAL}"  !=  ""  ]
        do
           #
           # extract next value from list
           #
```

```
            NEXT_LOV_VAL='echo "${VAL_VALUE}" | cut -d'|' -f${FIELD_NO}'
            if [ "${P_ARG_VAL}" = "${NEXT_LOV_VAL}"  ]
            then
                #
                # argument value matches one of the allowed values
                #
                return $TRUE
            else
                #
                # move pointer to the next allowed value
                #
                FIELD_NO='expr $FIELD_NO + 1'
            fi
        done
        #
        # processed all allowed values and a match is not found
        # invalid value provided for this argument
        #
        DisplayMessage  E  "${INVALID_VALUE}"  N
        return $FALSE
    #
    elif [  "${VAL_TYPE}"  =  "D"  ]
    then
        #
        # validate for directory existence
        #
        if [ !  -d  "${P_ARG_VAL}"  ]
        then
            DisplayMessage E "${DIR_NOT_EXISTS}"  N
            return $FALSE
        fi

    fi
fi
#
return $TRUE
#
}
#########################################################################
#    Name         :  ShowParameters
#    Overview     :   The function shows all the allowed parameters and
#                     their valid values
#    Notes        :
#########################################################################
ShowParameters ()
{
clear
echo "Usage:${SCRIPT_NAME}  <valid param>=<allowed value>"
echo "Parameter Description "
```

```
echo "====================  "
#
# loop around and select all arguments
#
for VAR in ${VALID_ARGUMENT_LIST}
do
   #
   L_ARG_NAME='echo "${VAR}" | cut -d':' -f1'
   VAL_REQ_IND='echo "${VAR}" | cut -d':' -f3'
   VAL_VALUE='echo "${VAR}" | cut -d':' -f5'
   #
   #
   if  [ "${VAL_REQ_IND}"  = "Y"  -a  "${VAL_TYPE}"  =  "LOV" ]
   then
      echo "parameter=${L_ARG_NAME} Valid Values=${VAL_VALUE}" | awk
{'printf("%-3Øs %-6s %-2Øs\n",$1,$2,$3)'}
   else
      echo "parameter=${L_ARG_NAME} Any valid value" | awk {'printf("%-
3Øs %-4s %-4s %-4s\n",$1,$2,$3,$4)'}

   fi
done
}
#########################################################################
#    Name      :  ParseCommandLine
#    Overview  :   The function parses the command line.
#
#     Notes    :  1.   The command line has the following
#                       options:
#
#                       graph=<graph_name>          mandatory
#                       graph_loc=<graph location>  optional
#                       log=<log directory>         optional
#                       error=<error directory>     optional
#                       reject=<reject directory>   optional
#                       sqltrace=<Y|N>              optional
#              ocf=<oracle configuration file>              optional
#              ocf_loc=<oracle configuration file location> optional
#              spdebug=<Y|N>                                optional
#              erbose=<Y|N>                                 optional
#########################################################################
ParseCommandLine ()
{
OPTION=1
while true
do
   #
   # select next argument=value pair from the command line $ARGL
   #
```

```
   ARGVAL='echo "${ARGL} " | cut -d' ' -f${OPTION}'
   #
   if [ "${ARGVAL}"  = "" ]
   then
      #
      # parsed the whole command line
      #
      break
   fi
   #
   # search for = sign in $ARGVAL
   #
   if   echo "${ARGVAL}"  |  grep "=" >  /dev/null 2>&1
   then
        :
   else
     DisplayMessage E "${INCORRECT_ARG_SPECIFICATION}"
     return $FALSE
   fi
   #
   # extract the argument name from $ARGVAL
   #
   ARG_NAME='echo ${ARGVAL} | cut -d'=' -f1'
   #
   # extract the argument value from $ARGVAL
   #
   ARG_VAL='echo ${ARGVAL} | cut -d'=' -f2'
   #
   #
   # validate argument
   #
   if  ValidateArgument  "${ARG_NAME}" "${ARG_VAL}"
   then
        :
   else
        return $FALSE
   fi
   #
   StoreArgumentValue "${ARG_NAME}" "${ARG_VAL}"
   #fi
   #
   OPTION='expr $OPTION + 1'
#
done
}
#########################################################################
#    Name       :  ReEvaluateVariables
#    Overview   :   The function resolves variable references.
#     Notes     :  1.
```

```
#####################################################################
ReEvaluateVariables  ()
{
TEMP_FILE='eval echo ${TEMP_FILE}'
#
# reevaluate log directory and log file
#
if  [  "${P_LOG_DIR}"  = ""  ]
then
    #
    #  log_dir parameter was not provided
    #
    SCRIPT_LOG_DIR='eval echo ${SCRIPT_LOG_DIR}'
else
    SCRIPT_LOG_DIR=${P_LOG_DIR}
fi
#
if  [  "${P_LOG_FILE}"  = ""  ]
then
    #
    #  log_file parameter was not provided
    #
    SCRIPT_LOG_FILE='eval  echo ${SCRIPT_LOG_FILE}'
else
    SCRIPT_LOG_FILE=${P_LOG_FILE}
fi
#
# re-evaluate error directory and error file
#
if  [  "${P_ERR_DIR}"  = ""  ]
then
    #
    #  err_dir parameter was not provided
    #
    SCRIPT_ERROR_DIR='eval eval   echo ${SCRIPT_ERROR_DIR}'
else
    SCRIPT_ERROR_DIR=${P_ERR_DIR}
fi
#
if  [  "${P_ERR_FILE}"  = ""  ]
then
    #
    #  log_file parameter was not provided
    #
    SCRIPT_ERROR_FILE='eval eval   echo ${SCRIPT_ERROR_FILE}'
else
    SCRIPT_ERROR_FILE=${P_ERR_FILE}
fi
#
```

```
}
#########################################################################
#    Name       :  InitialiseLogAndErrorFiles
#    Overview   :   The function initializes log and error files.
#     Notes     : 1. Files should be initialized with the appropriate
#                    headers.
#########################################################################
InitialiseLogAndErrorFiles ()
{
#
# initialize log file
#
( >  ${SCRIPT_LOG_DIR}/${SCRIPT_LOG_FILE} ) 2> ${TEMP_FILE}
if [ $?  -ne  Ø  ]
then
    #
    # initialization failed
    #
    ERR_MSG="Failed to initialize log file because "
    ERR_MSG="${ERR_MSG} 'cat ${TEMP_FILE} | sed s/'*'//'"
    DisplayMessage E "${OS_ERROR}"
    return $FALSE
fi
#
# initialize error file
#
( >  ${SCRIPT_ERROR_DIR}/${SCRIPT_ERROR_FILE} ) 2> ${TEMP_FILE}
if [ $?  -ne  Ø  ]
then
    #
    # initialization failed
    #
    ERR_MSG="Failed to initialsie error file because "
    ERR_MSG="${ERR_MSG} 'cat ${TEMP_FILE} | sed s/'*'//'"
    DisplayMessage E "${OS_ERROR}"
    return $FALSE
fi
#
return $TRUE
}
#########################################################################
#    Name         :   ProcessExit
#    Overview     :   The function removes all temporary files and
#                     exits with the exit code that is passed to the
#                     function.
#    Input        :   exit code
#    Notes        :
#########################################################################
ProcessExit ()
```

```
{
#
# assign parameter
#
P_EXIT_CODE=$1
#
rm -f ${TEMP_FILE}
#
exit ${P_EXIT_CIDE}
}
#########################################################################
#    Name         :   main
#    Overview     :   The function implements processing
#                     structure.
#     Notes       :
#########################################################################
main  ()
{
#
#  initialize  variables
#
InitialiseVariables
#
#  parse command line
#
if !   ParseCommandLine
then
           ProcessExit  $FEC
fi
#
#Re-evaluate variables
#
ReEvaluateVariables
#
#
if  !  InitialiseLogAndErrorFiles
then
        ProcessExit  $FEC
fi
#
ProcessExit  $SEC
}
#
#  package command line
#
SCRIPT_NAME=$Ø
ARGL="$@"
ARGC=$#
#  invoke  main
```

*Arif Zaman*
*DBA/Developer (UK)*


# Undeleting files in Unix

Since there is no undelete command in Unix, one way to provide that feature is to hide files to be deleted instead of deleting them immediately.

When the user types **rm**, the files to be deleted will be moved to a hidden directory located either in the user's home directory or in some other directory such as */tmp*. */tmp* may not be a good location if you are running skulker. In our environment we have a directory called */paultemp*, which is used to hold files and directories that are quasi temporary.

There are several ways to provide undelete functionality to Unix users. This article will demonstrate both a korn shell and a C shell implementation.

## KORN SHELL

Create a function in *.profile* called **del** or any other appropriate name, then create an alias to **rm** that will point to **del**.

The **del** function accepts an unlimited number of arguments, therefore the user can enter commands such as:

```
del file1 file2 file3
```

or:

```
del file*
```

etc.

The function first checks to see whether a trash directory exists. If there is no *.trash* directory for the user name, it creates one. It

then simply moves all the arguments (file names) passed to it to the user's hidden *.trash* directory.

An **undel** function should also be created in the *.profile*. This function does the opposite of **del**. It first checks that the arguments passed to it are valid file or directory names. If they are, it simply moves the files from the *.trash* directory to the pwd. Both the **del** and **undel** functions would have to be modified if one wants to move the files back to their directory of origin without having to explicitly **cd** there.

Finally, create a function in *.profile* called **clean**. This function will permanently delete files and directories in the *.trash* directory just like the empty recycle bin in Windows.

*.profile* code:

```
PATH=$PATH:/usr/local/bin
export PS1='hostname -s''[$PWD] > '
export EDITOR=/usr/bin/vi
set -o vi
errpt -a |more
clear
uptime
del()
{
  trashpath="/paultemp/.trash/"'whoami'
# create .trash directory if it does not exist
  if [ ! -d "$trashpath" ]
    then
      mkdir "$trashpath"
  fi
  while [ $# -gt Ø ]
  do
# make sure that argument is a valid file or directory before trying to
move
    if [ -f $1 -o -d $1 ]
       then
         mv $1 $trashpath
    fi
    shift
  done
return
}

undel()
{
```

```
#  set -xv
  trashpath="/paultemp/.trash/"'whoami'
  if [ ! -d $trashpath ]
     then
      echo No trash can was found.
      return
  fi
  while [ $# -gt Ø ]
  do
# make sure argument is a valid file or directory
        if [ -f $trashpath/"$1" -o -d $trashpath/"$1" ]
          then
# move file or directory from trash can to pwd
             mv $trashpath/"$1" .
        fi
        shift
  done
return
}


clean()
{
  trashpath="/paultemp/.trash/"'whoami'
  curdir='pwd'
  if [ ! -d $trashpath ]
     then
      echo No trash can was found.
      return
  fi

 cd $trashpath
# Explicit path to rm must be used here otherwise, the alias, del will
be used.
   ls $trashpath|xargs -I [] /usr/sbin/rm []
   cd $curdir
return
}

# Aliases

alias rm=del
```

## C SHELL IMPLEMENTATION

The C shell implementation is much shorter, but, of course, it is more cryptic. Here, two aliases, **del** and **undel**, should be placed

in *.login*. The **del** alias would be written as follows:

```
alias del mv \!:^ $HOME/.trash
```

The \ is needed to quote the !. \!:^ refers to the first word following the previous event. The previous event in this case would be **del** and the first word following **del** would the file name. Therefore:

```
del file1
```

would move *file1* to the *.trash* directory under the user's home directory.

Only one file can be deleted at a time with this implementation, but it will handle entire directory trees. Note that unlike the korn shell version, no filename or directory name validation is performed here.

The **undel** alias is:

```
alias undel mv $HOME/.trash/\!:^  .
```

This simply moves the file or directory from *.trash* to the current directory.

A clean alias has not yet been implemented for the C shell.

It is hoped that readers will improve on the above attempts.

*Anthony Richards*
*Unix System Administrator*
*Baltimore County Schools (USA)*



## The power of awk – again

Another helpful concept of **awk** is that you can compose a command, then make it react against a file.

For example, reading a file that is single spaced, then changing it to be double spaced would be accomplished with one simple **awk** command.

Input:

```
# cat the_report.txt
Line 1
Line 2
Line 3
#
```

Command:

```
awk '{print ; print ""}' the_report.txt > the_report_doublespaced.txt
```

Output:

```
# cat the_report_doublespaced.txt
Line 1

Line 2

Line 3

#
```

*David Miller*
*Database Architect*
*Baystate Health Systems (USA)*


# Displaying memory allocation

## INTRODUCTION

With the advent of 4GL languages, memory allocation has become an interesting topic because the software tends to be memory hungry. Quite often programs developed in these languages fail because of a lack of memory in the system, leaving system administrators baffled as to why – there being many gigabytes of real memory, and yet the program complains about lack of memory. Now the **ps** command can be used with its option **rss** (resident set size in kilobytes of real memory) and **vsz** (virtual set size in kilobytes) to trace the memory allocation for a given process and all its child processes. This article

contains code for a script called scamp.ksh (show current allocation of memory for a process). The script, for given process id, establishes all its child processes recursively, and then determines the rss and vsz for each of these processes. The **rss** option of the **ps** command shows the current allocation of real memory to a process and the **vsz** option displays the total allocation of memory (real plus virtual) to a process.

However, if the same program is invoked more than once, the text segment will be shared by all invoked instances of the program; therefore, for the purpose of this memory allocation calculation, we need to adjust the total memory allocation, and the script does just that. In theory, the total of rss for a process and its child processes should be the real memory requirement for that process at a specific point in time. The aforementioned script does exactly that (work out the total real memory allocation for a process and all its child processes at a specific point in time, which is when the script is running). This calculation cannot be taken as an authoritative guide for the memory requirements for a process, but should be treated as a guideline.

## SCAMP.KSH

```
#! /bin/
ksh##############################################################################
Name     : scamp.ksh (show current allocation of memory for a
#                           process)
#
#   Overview : The script shows allocation of memory for a specific
#              process and all its child processes at a given point
#              in time.
#
#   Notes    : 1. The script contains following functions:
#                       o InitialiseVariables
#                       o InitialiseReportFile
#                       o HandleInterrupt
#                       o PrintReport
#                       o MoveCursor
#                       o DisplayMessage
#                       o FormatUnderscores
#                       o FormatReportLine
#                       o ProcessExit
#                       o main
```

```
#                     o PerformTestForBackGroundJob
#                     o HandleTerminalIoInBackgroundInterrupt
#                     o GetProcessId
#                     o BuildChildProcessTree
#                     o ListAllProcesses
#                     o GetAllChildProcessIds
#                     o CalculateMemoryAllocatedForAllProcesses
#                     o ViewReport
#
#            2.  The script generates a report called
#                scamp_$$.rep in /tmp directory
#
#            3.  A quick way to familiarize yourself with the script
#                is to execute the script as follows:
#                scamp.ksh help=y  > /tmp/help.dat and read the
#                help text
#
#            4.  The script requires root access in order to
#                execute bootinfo program to get the information
#                regarding the amount of physical memory in the
#                system.
#
#   History  :
#   Date      Author    Description
#########################################################################

#########################################################################
#   Name     : InitialiseVariables
#   Overview : The function initializes all required variables.
#   Notes    :
#########################################################################
InitialiseVariables()
{
export SELECTED_PID=Ø # process id for which memory allocation is to be
displayed
export SEARCH_PID=Ø   # used in GetAllChildProcessIds ()
#
STDOUT="1"
STDERR="2"
#
integer DUMMY_INT
# temporary files
export TEMP_FILE_1="/tmp/scamp_$$_1.tmp"
export TEMP_FILE_2="/tmp/scamp_$$_2.tmp"
export TEMP_FILE_3="/tmp/scamp_$$_3.tmp"
export REPORT_FILE="/tmp/scamp_$$.rep"
# file containing formatted process ids
#
export PROC_FILE=/tmp/scamp_$$.proc
# escape  sequences
```

```
#
ESC="\ØØ33["
RVON=[7m                    # reverse video on
RVOFF=[27m                  # reverse video off
BOLDON=[1m                  # bold on
BOLDOFF=[22m                # bold off
BON=[5m                     # blinking on
BOFF=[25m                   # blinking off
# define exit codes
#
export SEC=Ø
export FEC=1
# function return codes
#
export TRUE=Ø
export FALSE=1
# date and time
#
DATETIME='date "+%d/%m/%Y at %H:%M:%S"'
SLEEP_DURATION=4        #  no of seconds allowed for sleep command
ERROR="scamp.ksh:ERROR:"
INFO="scamp.ksh:INFO:"
#  message
#
WORKING="Working"
INTERRUPT="Program interrupted ! Quitting early"
INVALID_OPTION="Invalid entry"
PRINT_OK="Successfully submitted the print job"
PRINT_NOT_OK="Failed to submit the print job"
INVALID_ENTRY="Invalid entry"
NOT_NUMERIC="Value must be numeric${RVOFF}"
OSERROR="\${ERR_MSG}"
INVALID_PID="The process \(\${SELECTED_PID}\) currently does not exist"
NULL_PID="Must provide a process id"
FORMATTING_PROCESS_IDS="Formatting process ids"
INVALID_ARG_SPECIFIER="Invalid argument specifier for argument, \${ARG}"
INVALID_VALUE="Invalid value for argument, \${ARG_TYPE}"
POS_INT_REQ="An integer greater than Ø is required for argument,
\${ARG_TYPE}"
INVALID_ARG_TYPE="\${ARG_TYPE}, is an invalid argument type"
DUPLICATE_ARG_TYPE="Duplicate argument, \${ARG_TYPE}"
MANDATORY_PID="Must provide a process id for script running in batch
mode"
# define  signals
#
SIGNEXIT=Ø  ;  export SIGNEXIT   # normal exit
SIGHUP=1    ;  export SIGHUP     # when session disconnected
SIGINT=2    ;  export SIGINT     # ctrl-c
SIGTERM=15  ;  export SIGTERM    # kill  command
}
```

```
#########################################################################
#  Name     : HandleInterrupt
#  Overview : The function calls ProcessExit.
#  Input    :
#  Notes    :
#########################################################################
HandleInterrupt ()
{
DisplayMessage I "${INTERRUPT}"
ProcessExit  $FEC
}
#########################################################################
#    Name     : MoveCursor
#    Input    : Y and X coordinates
#    Returns  : None
#    Overview : It moves the cursor to the required location (Y,X).
#    Notes    : 1. It must be run in ksh for print to work. Also,
#                  print must be used to move the cursor because echo
#                  does not seem to work.
#########################################################################
MoveCursor  ( )
{
trap "HandleInterrupt " $SIGINT  $SIGTERM $SIGHUP
YCOR=$1
XCOR=$2
 echo     "${ESC}${YCOR};${XCOR}H"
}
#########################################################################
#  Name     : DisplayMessage
#  Overview : The function displays message
#  Input    : 1. Message type (E = Error, I = Informative)
#             2. Error Code as defined in DefineMessages ().
#             3. Message to be acknowledged flag (Y=yes N=no)
#  Notes    :
#
#########################################################################
DisplayMessage ( )
{
MESSAGE_TYPE=$1
MESSAGE_TEXT='eval echo $2'
ACKNOWLEDGE_FLAG="$3"
L_ERROR='eval eval   echo ${ERROR}'
L_INFO='eval  eval echo ${INFO}'
# default the message acknowledge flag
#
if [ "${ACKNOWLEDGE_FLAG}" = ""  ]
then
  ACKNOWLEDGE_FLAG="N"
fi
# add message acknowledgement indicator
```

```
#
if [ "${ACKNOWLEDGE_FLAG}" = "N"  ]
then
    MESSAGE_TEXT="${MESSAGE_TEXT}..."
fi
#
#RightPadMessageText "${MESSAGE_TEXT}"
#
clear
MoveCursor 24 1
#
if [ "${MESSAGE_TYPE}" = "E" ]
then
     echo  "${RVON}${L_ERROR}${MESSAGE_TEXT}${RVOFF}\c"
else
     echo  "${RVON}${L_INFO}${MESSAGE_TEXT}${RVOFF}\c"
fi
#
# examine message acknowledge flag
#
if [ "${MODE}" = "B"  ]
then
     return ${TRUE}
fi
#
if [ "${ACKNOWLEDGE_FLAG}"  =  "Y"  ]
then
      read  DUMMY
else
      sleep  "${SLEEP_DURATION}"
fi
return ${TRUE}
}
#########################################################################
#  Name     :FormatReportLine
#  Overview : The function reformats a report line for its command
#             text to be printed in a wrap-around manner within a
#             fixed-width column.
#  Notes    :
#########################################################################
FormatReportLine ()
{
# assign parameter
#
P_LINE="$1"
#
NO_CHARS='echo "${P_LINE}\c" |  wc -c'
FIRST_EIGHTY_CHARS='echo "${P_LINE}" | cut -c1-8Ø'
REST_OF_CHARS='expr "${P_LINE}" | cut -c81-'
#
```

```
REPORT_LINE="${FIRST_EIGHTY_CHARS}\n"
#
SEP1_LENGTH='echo "${P_LINE}" | cut -d'=' -f1 |  wc  -c'
SEP2_LENGTH='echo "${P_LINE}" | cut -d'=' -f2 |  wc  -c'
SEP3_LENGTH='echo "${P_LINE}" | cut -d'=' -f3 |  wc  -c'
SEP4_LENGTH='echo "${P_LINE}" | cut -d'=' -f4 |  wc  -c'
TOTAL_SPACES='expr $SEP1_LENGTH + $SEP2_LENGTH + $SEP3_LENGTH +
$SEP4_LENGTH'
#
IND=Ø
while [ $IND -lt $TOTAL_SPACES  ]
do
  REPORT_LINE="${REPORT_LINE} "
  IND='expr $IND + 1'
done
#
REPORT_LINE="${REPORT_LINE}${REST_OF_CHARS}"
}
#########################################################################
#  Name     : FormatUnderscores
#  Overview : The function assigns appropriate number of
#             underscores(=) to the variable UNDERSCORE to be used
#             in conjunction with a header.
#  Input    : Line containing  the  header
#  Notes    :
#########################################################################
FormatUnderscores ()
{
# assign parameter
#
LINE="$1"
# initialize  UNDERSCORE
#
UNDERSCORE=
# initialize  index
#
IND=1
#  get no of characters in  $LINE
#
NO_CHARS='echo "$LINE" |  wc  -c'
#  subtract the carriage return
#
NO_CHARS='expr $NO_CHARS - 1'
while [ "$IND" -le "$NO_CHARS" ]
do
  UNDERSCORE="${UNDERSCORE}="
  IND='expr $IND + 1'
done
}
#########################################################################
```

```
# Name        : InitialiseReportFile
# Description : The function initializes the report file
# Notes       :
#########################################################################
InitialiseReportFile ()
{
HEADER="Memory Allocation (in bytes) for Pid($SELECTED_PID) and Child
Processes"
FormatUnderscores "${HEADER}"
echo "${HEADER}"      >  ${REPORT_FILE}
echo "${UNDERSCORE}"  >> ${REPORT_FILE}
echo "Mem=Ø indicates that the process is sleeping and not resident in
memory" >> ${REPORT_FILE}
echo "Rmem= indicates the amount of real memory the process has
acquired" >> ${REPORT_FILE}
echo "Vmem= indicates the amount of total (real + virtual ) memory the
process has acquired" >> ${REPORT_FILE}
echo "Total memory allocation is adjusted for multiple invokation of
same program"  >> ${REPORT_FILE}
echo "by adding the memory allocation for text segment once for that
program\n"   >> ${REPORT_FILE}
}
#########################################################################
# Name        : PrintReport
# Description : It conditionally prints the report file.
# Notes       :
#########################################################################
PrintReport ( )
{
# print file
#
while  true
do
  clear
  echo "Do you wish to print the output file(Y/N)?:\c"
  read REPLY
  case  $REPLY   in
      n|N ) return $TRUE ;;
     y|Y  ) break ;;
      *   )   : ;;
  esac
done
#  get printer name
#
while  true
do
   clear
   echo "Enter printer name for lp command(q to quit):\c"
   read  PRINTER
   case  $PRINTER in
```

```
      "" )   : ;;
      q|Q)  break ;;
       * ) lp  -d${PRINTER} ${REPORT_FILE}  > ${TEMP_FILE_1}  2>&1 ;
           if [  $?  -eq  0  ]
           then
               DisplayMessage I  "${PRINT_OK}" ;
               break ;
           else
               ERR_MSG='cat ${TEMP_FILE_1}'
               DisplayMessage  E  "${OSERROR}" ;
           fi ;;
   esac
done
}
```

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

*Arif Zaman*
*DBA/Developer (UK)*



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 AIX users have written to make their life, or the lives of their users, easier. We are also looking for hints and tips from experienced users.

We will publish it (after vetting by our expert panel) and send you a cheque when the article is published. Articles can be of any length and can be e-mailed to Trevor Eddolls at trevore@xephon.com.

A free copy of our *Notes for contributors*, which includes information on our payment rates, is available from our Web site at www.xephon.com/nfc.

# AIX news

Sybari Software has announced that Antigen, its anti-virus software, now runs on Lotus Domino servers running on IBM's AIX and Sun Solaris Operating Systems.

Antigen provides messaging security for Domino Servers. As well as virus protection, it provides customer-driven features such as worm purge.

Sybari's Antigen for Domino offers defence against propagating e-mail viruses, worms, and malicious code. Antigen is a server-level anti-virus solution that delivers protection through its multiple scan engine approach, as well as content-filtering capabilities.

For further information contact:
Sybari Software, 353 Larkfield Rd, East Northport, NY 11731, USA.
Tel: (631) 630 8500.
URL: http://www.sybari.com/products/antigen_notes.asp.

* * *

IBM has issued an Autonomic Computing toolkit to help developers create self-managing characteristics to be used in data centres or other IT environments where on-demand or utility computing may be used.

Written with the Java-based open-source Eclipse platform, the kit is designed to help developers add autonomic computing to their applications with the IBM Software Development Platform, an application development environment.

The kit consists of several tools IBM software engineers created to help a network of servers manage and regulate themselves so that IT administrators may be free to address other IT issues. This includes run-times, tools, usage scenarios, and documentation that correspond with IBM's Autonomic Computing Blueprint.

The toolkit, which supports AIX, Linux on Intel systems, and Windows 2000, can be found on the company's Autonomic Computing Web site.

For further information contact your local IBM representative.
URL: http://researchweb.watson.ibm.com/autonomic/overview/solution.html.

* * *

Enea Embedded Technology has announced Version 5 of Polyhedra, its RDBMS.

Polyhedra 5.0 has two major enhancements: Unicode support allows text in the database to include accented characters and special symbols; while with the schema migration support, customers can field-upgrade their Polyhedra-based applications with minimal disruption.

Release 5.0 supports major database standards (eg SQL, ODBC, and JDBC) and provides cross-platform support for multiple operating systems (eg Windows, Linux, Solaris, IRIX, AIX, OSE, and VxWorks).

For further information contact:
Enea Embedded Technology, 12760 High Bluff Drive, San Diego, CA 92130, USA.
Tel: (858) 720 9958.
URL: http://www.ose.com/products/product.php?product_id=180.