109

AIX update

November 2004

In this issue

A tcl 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

Bob Thomas
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.

Disclaimer

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

Contributions

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

This issue is dedicated to the memory of Chris Bunyan, co-founder of Xephon and creator of the *Update* journals.

Printed in England.

Understanding the strings command and NLS

In my job providing National Language Support (NLS) to our organization, I need to extract strings of text from program files and send them to various countries to be translated into their respective languages. When the text returns, the next step is to integrate this translated text back into the source code to be built into the National Language Versions of the product.

Isolating translatable text from source code developed with a text editor can be simple. All the text and code are visible as printable characters viewable from any text editor. However, sometimes you may be called upon to determine whether there are text strings to be found in binary files, and that can be much more difficult.

In some cases, you may possess only binary files without having the source code from which the files were compiled. Certain proprietary editors may produce binary output without giving you the ability to export into flat text files.

AIX offers a command that can help – the **strings** command.

The **strings** command can search a binary file and report occurrences of text strings found. If you perform NLS activities, you may find the **strings** command quite handy. However, you need not be involved with NLS to benefit from my experiences with the **strings** command. The examples I give on how we use **strings** can be applied anywhere you would find the need to locate and process text strings within a binary file.

The command looks for strings of four or more printable characters ending with the newline or the null character. There is a flag for specifying alternative character lengths if needed. The syntax of the **strings** command is shown below.

SYNTAX

Typical syntax for the **strings** command is:

```
strings [ -a ] [ -n num ] [ -t offset ] [ File ... ]
```

where:

- -a – tells **strings** to search the entire file rather than just the data section.
- -n num – tells **strings** to use a length other than 4 to determine a valid string.
- -t offset – precedes each string by its offset into the file. Valid offset designators are **d** for decimal, **o** for octal, and **x** for hexadecimal.
- File – the file to be searched.

PROBLEMS WITH BINARY FILES

In theory, you could edit a binary file with a text editor and seek printable lines of text. In practice, that can be difficult for the following reasons:

1 You may not be able to see all the text without scrolling a long way across the screen left and right because some editors present binary files using lines each containing 4,000 or more characters.

2 Formatting may be difficult to inspect for fixed line lengths because the editor may not properly present line end characters. See below for an example of 'barber pole' formatting where the plus character (+) represents non-printable binary characters.

```
+++This is an example of sentence one in a ++++++++++++
+++++binary file containing fixed-length     ++++++++++
+++++++records.  Here is an example of sentence++++++++
+++++++++two in a binary file containing fixed-  ++++++
+++++++++++length records.                         ++++
```

3 Formatting may also be difficult for variable length lines for the same reason. See below for an example:

```
++++This sentence has 32 characters.++++This has only
17.++++However, this sentence has many more than the first - 68
characters.++++But this one has just 25.++++
```

USING THE STRINGS COMMAND

Let’s see how the strings command works. The example below represents a sample binary file one might encounter. For the purpose of this example, assume that the plus sign (+) is a binary non-printable character. Further, assume that all the data below would appear concatenated onto one line if viewed using a text editor.

```
Name of Student     Grade   Track   Rating   Teacher++++--------------
----   ----   ------   --------------++++Hill, Arthur          5
IV3.8     Mr Franklin++++Marquez, Maria        4      III     4.0
Srta Trinez++++Vick, Hector          5      II      2.6     Mrs
Grapple++++Salcov, Dmitri        3      I       3.4     Mr
White++++Lewis, Sylvia         2      III   3.8     Mrs Muscovitz++++
```

Assuming the file is named *school.record*, the example below is what you would get if you entered the command:

```
strings school.record
```

Note that now all columns properly line up and all non-printable characters are stripped.

```
Name of Student     Grade   Track   Rating   Teacher
---------------     -----   -----   ------   --------------
Hill, Arthur          5      IV      3.8     Mr Franklin
Marquez, Maria        4      III     4.0     Srta Trinez
Vick, Hector          5      II      2.6     Mrs Grapple
Salcov, Dmitri        3      I       3.4     Mr White
Lewis, Sylvia         2      III     3.8     Mrs Muscovitz
```

OFFSETS

The example below shows what the first column of the output file would look like if the offset flag (-t) were used as follows:

```
strings -t d school.record   (for decimal offset)
strings -t x school.record   (for hexadecimal offset)
strings -t o school.record   (for octal offset)
```

Sample output formats with the offsets specified appear thus:

```
       DECIMAL                  HEXADECIMAL              OCTAL


   0 Name of Student       0 Name of Student       0 Name of Student
  56 --------------       38 --------------       70 ---------------
```

```
119 Hill, Arthur          77 Hill, Arthur         167 Hill, Arthur
179 Marquez, Maria        b3 Marquez, Maria       263 Marquez, Maria
239 Vick, Hector          ef Vick, Hector         357 Vick, Hector
299 Salcov, Dmitri       12b Salcov, Dmitri       453 Salcov, Dmitri
356 Lewis, Sylvia        164 Lewis, Sylvia        544 Lewis, Sylvia
```

STRINGS AND NLS

Our NLS group uses strings for many tasks:

1 To determine whether binary files contain translatable strings.

2 To extract the strings from binary files to send for translation.

3 To seek specific strings in binary files.

4 To inspect the general formatting of the text.

5 To isolate the text to perform pre-translation tests.

6 To check the returned (translated) text for errors.

Let’s take a closer look at each, to see how you could use the **strings** command for similar tasks.

Determine whether binary files contain translatable strings

Before we can create a translation strategy, we need to find out whether any of the source files contain text that is printable, displayable, or otherwise viewable by an end user. Executing **strings** against the binary source files would yield all printable strings in all files. Those containing no printable strings can be ignored for translation. Of the remaining, decisions would need to be made on whether the strings would appear in a user interface, and therefore would need to be translated. A script can be written that redirects the output of the **strings** command to a dummy file and just displays the names of files containing printable strings, if all you need is a list of files.

The **strings** command is also useful for detecting files that might have missed being previously identified as containing translatable text. For example, if you were to run **strings** against a directory of the latest program files and compare the results with a list of files previously sent for translation, you might discover new files that were missed.

Extract the strings from binary files to send for translation

Under certain circumstances, it may be desirable or necessary to use **strings** to actually extract the translatable text to send for immediate translation. An example would be if your development teams have a mechanism for merging the returned translations back into the binary file or if you need to begin the translation process before they have developed such a mechanism.

Seek specific strings in binary files

You may need to seek specific strings in a binary file. You cannot use the AIX **grep** command on a binary file because **grep** may consider the entire binary file as one line, and you may simply get the entire file as a result. In any case, running **grep** against a binary file would give unpredictable results.

However, you can pipe the output of the strings command into **grep** to get your results. Here is a sample command for running **strings** and **grep** against a binary file:

```
strings <filename> | grep 'specific text'
```

Below are the results of issuing the following **grep** command against the sample file:

```
strings school.record | grep III


Marquez, Maria         4       III     4.Ø     Srta Trinez
Lewis, Sylvia          2       III     3.8     Mrs Muscovitz
```

Inspect the general formatting of the text

Sometimes you may just want to look at the printable strings in a binary file without actually processing the text in any way. For example, inspecting the results may be useful in formulating a pre-translation strategy, helping you to determine how you want to prepare the text for translation prior to sending the text to the countries.

Also, the results of executing the **strings** command might expose:

1 Formatting issues such as columns not properly lining up. This would be very difficult to detect if you observed text such as in the earlier examples using an ordinary text editor. However, you can see that the formatting is much easier to inspect after the **strings** command is run.

2 Placement problems such as finding text that is obviously physically in the wrong part of the file. Finding text near the bottom of a file when it should be in the middle is much easier when the data is presented in a readable format rather than in three lines each containing more than 4,000 characters.

3 Translation enablement issues, such as hard-coded links in the source language or fields not having been properly sized to allow for translation expansion, may result in text truncations in the final file.

Isolate the text to perform pre-translation tests

Here are some examples of using the **strings** command to pre-process the text in preparation for sending to the countries for translation.

1 *Word counts*. Translation planners often need to know how many words will be translated. By piping the output of the **strings** command into **wc -w** (AIX's word count command) you can display the number of printable words in a binary file.

Entering **strings school.record | wc -w** against the sample file would yield a word count of 47.

2 *Line counts*. In the event you need to determine the number

of translatable lines in the binary file, pipe the output to **wc -l**.

Entering **strings school.record | wc -l** against the sample file would yield a line count of 7.

3 *Import/export*. You can import the output of strings into a word processor to perform spell checks, reviews, and terminology consistency validations of the source text before sending to other countries.

Check the returned (translated) text for errors

In the event that the translating countries have the necessary authoring software to directly produce translated output in binary format, you may need to use the **strings** command to inspect their results. For example, you might need to run **strings** against the source language and all the target languages so your test group can do side-by-side validations of the translations, or check for corrupted national characters, formatting issues of the translated text, or other problems with the returned text.

SUMMARY

Though the AIX **strings** command has many uses in National Language Support, you should be able to find similar uses whenever you need to inspect or process printable strings in binary files.

David Chakmakian
Software Engineer (USA)

Show Unix errors

INTRODUCTION

Sue (show Unix error) is a small C program that displays Unix system error message(s). On various occasions, Unix applications may return a code (or error number) without displaying the full error message. The **sue** executable can be used to look up the corresponding error message. When **sue** is executed with a specific error number, it prints out on stdout the corresponding error message; otherwise, it will print all the defined error numbers with their corresponding error messages on stdout.

COMPILING SUE

Use the following command to compile **sue**:

```
cc   sue.c  -o  sue
```

LISTING

```
/************************************************************************
*   Name     : sue (show Unix error)
*  Overview : The script displays Unix error message for a given error
*             number.
*             If the error number is not specified, it displays all
*             error messages
*  Input    : error number
*  Notes    :
***********************************************************************/
/***********************************************************************
*                             Include Files                            *
***********************************************************************/
#include  <errno.h>
#include  <stdio.h>
#include  <string.h>
#include  <stdlib.h>
/**********************************************************************
*                        Function Proto Types                         *
**********************************************************************/
int main ( int *argc ,  char *argv[]) ;
int  ParseandValidateCommandLine (int *argc, char **argv) ;
/***********************************************************************
*                            Module Messages                           *
***********************************************************************/
#define  USAGE   "Usage:sue <error no> | sue"
#define  WRONG_ARG_COUNT   "Error:Invalid argument"
#define  INVALID_NO        "Error:Invalid error no"
#define  MAX_ERR_NO        151
/**********************************************************************
*                             Module Constant                         *
**********************************************************************/
/*
 * function return codes
 */
#define  FAILURE 0
#define  SUCCESS 1
/*
 * exit codes
 */
#define SEC  0
#define FEC  1
/*
 * global
 */
int  p_errno = -1 ;
/***********************************************************************
*  Name     : main
*  Overview : Function main ()
*  Input    : Error Number ( optional )
*  Notes    :
**********************************************************************/
int  main ( int *argc ,  char *argv[])
{
  char  *err_msg ;
  int  i ;
/*
 * parse comamnd line
 */
if  ( ParseAndValidateCommandLine (argc, argv) == FAILURE )
  exit (FEC) ;
/*
 * check for specified error numnber
 */
if ( p_errno !=  -1 )
 {
  err_msg = (char *) strerror(p_errno);
  printf("%s\n", err_msg );
  return (SEC );
 }
/*
```

```
 * print out all message
 for ( i = 0 ; i < 152 ; i++  )
 */
 for ( i = 0 ; i < MAX_ERR_NO + 1  ; i++  )
  {
   err_msg = ( char *) malloc(200);
   err_msg = (char *) strerror( i );
   printf("Error No=%d Error Msg=%s\n",i, err_msg );
  }
return (SEC );
}
/***********************************************************************
*  Name     :  ParseAndValidateCommandLine
*  Overview : The function parses the command line.
*  Notes    :
***********************************************************************/
int  ParseAndValidateCommandLine (int  *arg_count,  char *arg_val[])
{
int   errno  ;
char  *chr_errno; /* string containing  converted error no */
/*
 * check for argument count
 */
 if (  arg_count >  (int *)     2  )
  {
   printf("%s\n", WRONG_ARG_COUNT );
   printf("%s\n", USAGE );
   return ( FAILURE);
  }
/*
 * check for valid number ( between 0 and 255 )
 */
 if (  arg_count >  (int *)     1  )
  {
    errno =  atoi(arg_val[1]);
    /*
     * print converted error number into a string
     */
    chr_errno=malloc(500);
    sprintf(chr_errno,"%d",errno);
    /*
     * compare converted error number string
     *  with original input string containing error number
     */
    if ( errno == 0  &&  strcmp(chr_errno, arg_val[1]) == 0 )
     {
       /*
        * input string must contain numeric 0
        */
        p_errno = errno ;
     }
    else
      {
       /*
        *  compare converted error number range
        */
       if (  errno <1 || errno > MAX_ERR_NO )
        {
          printf("%s\n", INVALID_NO );
          printf("%s\n", USAGE );
          return ( FAILURE);
        }
      else
        p_errno = errno ;
     }
 }
return ( SUCCESS);
}
```

Arif Zaman
DBA/Developer (UK)

A short introduction to simultaneous multi-threading and shared processor partitions with AIX 5.3 and POWER5

INTRODUCTION

The recently announced POWER5 servers from Bull and IBM include a number of important enhancements over the previous generation of machines. Of these, perhaps the three most remarkable are:

- The virtualization of disks and networks.
- Shared processor partitions or micro-partitions.
- Simultaneous multi-threading.

Virtualization of disks and networks allows a single hardware resource to be shared in a controlled manner between several partitions or operating systems, and these operating systems may be AIX, Linux, or OS/400.

Shared processor partitioning or micro-partitioning is similar to virtualization but applied to the POWER5 CPUs. It allows a number of partitions or operating systems to share a common pool of microprocessors.

Simultaneous multi-threading is a feature of the POWER5 processor whereby a number of CPU resources are doubled to allow the concurrent running of two executable threads on a single microprocessor core.

This article provides a short introduction to the latter two technologies.

SHARED PROCESSOR PARTITIONS

The notion of a partitionable machine was introduced in AIX 5.1 on POWER4 hardware. Here a single server is split into a number of discrete logical partitions, each partition running its own operating system (AIX or Linux). This was called logical partitioning or LPAR. AIX 5.2 improved on AIX 5.1 by allowing computing resources to be added and removed from partitions without requiring a shutdown of the partition, ie dynamic logical partitioning or DLPAR. On these systems the minimum allowable system resources are 1 CPU, 256MB of memory, and an I/O slot with a disk attached. The granularity of DLPAR operations is a single processor (even though there are two CPU cores on each POWER4 chip), so a 32-processor PL3200 may have up to 32 independent partitions running either AIX or Linux. The partitioning of the machine is managed by a low-level software layer known as the hypervisor. The hypervisor ensures that each partition is isolated and has access only to the resources that have been allocated to it – a partition has no knowledge or visibility of any other partitions running on the same platform.

AIX 5.3 on POWER5 takes partitioning a step further. Whilst it is still possible to have partitions that operate in the same manner as in AIX 5.2 a new sort of partition has been introduced called a shared processor partition (SPLPAR) or micro-partition. Physical processors in the new servers are placed into one of three pools: one for dedicated processor partitions (as available in AIX 5.2 on POWER4), one for micro-partitions, and one for off-line or spare processors.

Note: AIX 5.3 does not include the notion of affinity partitions that are available in AIX 5.2.

The dedicated processor pool contains those processors that will work for just one partition. The dedicated pool may host several partitions, but each individual processor is dedicated to just one partition. Thus, the granularity of CPUs for a dedicated processor partition is a single processor – a 32-way platform may have up to 32 dedicated processor partitions, each with a single CPU.

The spare pool contains physical processors that are not available to the partitions. It is used for non-disruptive upgrades in the 'Power On Demand' programme, or for the dynamic replacement of a failing processor in one of the other two pools.

The shared processor pool contains all those physical processors that are not in either of the other two pools. The processors in this pool are shared amongst all the shared processor partitions (micro-partitions). A micro-partition may run on any or all of the processors in the shared processor pool.

Micro-partitions run not on physical processors but on virtual ones. A micro-partition must comprise at least one virtual processor. The 'size' of a virtual processor is between 10% and 100% of a physical processor, ie each physical processor may host up to 10 virtual processors. This sharing of processors is managed by the hypervisor, which time-slices each virtual processor on any of the physical processors in the shared processor pool. Thus it is possible to have 10 times as many partitions in a server as there are physical processors, with an upper limit of 254 active partitions per system.

When defining a partition, the administrator stipulates the amount, or share, of CPU resources that will be guaranteed to be made available to the partition and the number of virtual processors on to which this entitlement will be mapped. The amount of CPU allocated to a partition can never exceed the number of processors in the shared pool. The processor entitlement is expressed as a number of processing units. For example, if four partitions equally share two physical processors, then each partition would have an entitlement of 0.5 CPUs. The entitlement is sometimes expressed in terms of processing units. There are 100 processing units per physical processor.

The number of virtual processors seen by a partition does not have to have a one-to-one mapping with the number of physical processors to which it is entitled.

The partition's entitlement is spread across its virtual processors. Consider the example above, with four partitions each with an entitlement of 0.5 CPUs. If one partition has two virtual CPUs, then each virtual CPU would be 'worth' 0.25 physical CPUs. If another partition were created with 10 virtual CPUs, then each virtual CPU would be worth just 5% of a physical CPU. These figures indicate the average virtual processor size. It is possible for a high-priority single-threaded application running on one of the virtual processors to consume all the partition's entitlement for a period of time. In this case the single virtual CPU would have the power of half a physical processor and the other virtual processors would be idle.

It should be noted that though it is possible for the number of virtual CPUs to be different from the number of physical CPUs, it is not always a good idea. The computing power of your partition is determined by its entitlement not by the number of virtual processors. As a rule-of-thumb, provide your micro-partition with a number of processors equal to the partition's entitlement rounded up to the next whole number.

One side effect of micro-partitioning is that since each virtual CPU is time-sliced onto a physical processor, the time each virtual CPU executes is less than wall clock time. This has a number of important implications, most notably for performance monitoring tools such as iostat, vmstat, topas, and the AIX Performance Toolbox. Note too that there is a new command, lparstat, that will give the current status of the partition.

Capped and uncapped partitions

As seen above, a partition's entitlement will be guaranteed to be available should it be required. However, should a partition not be able to make use of all its entitlement because of a low workload or all applications being in an un-runable state, then it can donate, or cede, its unused cycles to any of the other partitions in the same server that have high workloads. This too has an impact on the performance tools, because now if a partition is using more CPU than its entitlement, the CPU load is more than 100%. In particular the interpretation of I/O wait time versus idle time becomes difficult. There are additional columns in the performance tool's output to show the CPU usage relative to a partition's entitlement.

When defining and creating your partition, you can specify whether you wish it to make use of any unused cycles from other partitions, and indicate a relative priority in the picking order. A partition that is allowed to pick up spare cycles is called an 'uncapped partition' whereas a partition that can use only its defined entitlement and no more is called a 'capped partition'. By default a partition is created in uncapped mode. Since an uncapped partition may consume more CPU than its entitlement, the percentage of CPU busy time can exceed 100%.

SIMULTANEOUS MULTI-THREADING (SMT)

Simultaneous Multi-Threading (SMT) is available on POWER5-based servers – it is similar to the hyperthreading technology found on some 32-bit Intel processors. The POWER5 processor has special hardware to allow it to maintain two execution contexts. Thus two threads can execute concurrently on a single processor – each thread makes use of the execution units unused by the other. Note that each POWER5 chip contains two physical processors or cores and SMT is enabled for each core, thus a single POWER5 chip may execute up to four threads.

SMT of Single-Thread (ST) execution mode is controlled by AIX. By default it is enabled at boot time. Enabling SMT effectively doubles the number of processors seen by AIX, thus on a four-processor machine with SMT, AIX (and commands such as **bindprocessor –q**) will see (and schedule on) eight CPUs. The processors seen by AIX with SMT enabled are called ‘logical processors’. The new **smtctl** command without any parameters shows the current SMT state and shows how the logical processors are bound to virtual or physical ones (depending on whether the processors are dedicated or shared).

SMT can be switched on or off at any time without having to shut down the server or the partition; indeed AIX will automatically shut down secondary threads when CPU load is low. The default SMT state at boot can be configured using SMIT or WebSM, or from the command line with the **smtctl** command.

The performance gain witnessed using SMT depends on the nature of the running application, but it is not unusual to see a 30% or 40% improvement in execution time with SMT enabled at 100% CPU load. Note that SMT is not possible in partitions running AIX 5.2. A single threaded application will almost certainly run faster in ST mode than SMT mode because all the processor’s resources are dedicated to the executing program. SMT offers a benefit for multi-threaded applications or when running several applications concurrently.

Virtual processors and SMT

As discussed above, simultaneous multi-threading is controlled by the partition and not by the physical processor. Thus it is possible to have two partitions sharing the same physical processor, one with SMT enabled and the other with it disabled.

The POWER5 CPU will be switched to the correct mode for each partition.

Further, in shared processor partitions AIX only ‘sees’ the virtual processors, thus SMT is enabled on virtual CPUs. With SMT enabled, the AIX scheduler will see a number of logical CPUs that is twice the number of virtual CPUs (which, you will remember, is not necessarily the same as the number of physical ones).

DYNAMIC PARTITIONING OPERATIONS WITH SMT AND MICRO-PARTITIONS

Using the Hardware Management Console (HMC), it is possible to add, move, or remove virtual processors to and from partitions and to change a partition’s processor entitlement dynamically (ie without rebooting the partition) in the same way as it is possible to move physical processors between partitions on POWER4 partitionable servers running AIX 5.2

CONCLUSION

Micro, or shared processor, partitions, along with simultaneous multi-threading, provide a greater degree of flexibility and adaptability than is obtainable with the dynamic logical partitioning available with POWER4 and AIX 5.2. This flexibility is most likely to benefit mixed workloads such as consolidation projects or to allow a Java-based 3-tier infrastructure (Web server, application server, and database) to be hosted on a single platform. At the same time this flexibility introduces a certain degree of complexity, in particular for the sizing of partitions and for some of the metrics reported by the performance monitoring tools.

Jez Wain
System Architect (France)

Finding hosts that cannot be pinged

The following script can be used to find problems in */etc/exports*. The script lists any hosts from */etc/exports* that cannot be pinged. Those host names should be deleted from */etc/exports*.

SCRIPT

```
[kaiserro@SPCWS]/u/kaiserro/script > vi exnodes
#!/usr/bin/ksh
# Script extracts node names from /etc/exports and tries to ping them
# If ping fails the node is listed.
cat /etc/exports | while read x
        do
                for i in $(print  - $x|sed -e "s/:/ /g"|sed -e "s/,/ /g"
|sed -e "s/=/ /g")
                do
                        vHost=$(print  - $i|grep -v "^/")
                        case "${vHost}" in
                                '-ro')
                                        ;;
                                '-root')
                                        ;;
                                'root')
                                        ;;
                                'access')
                                        ;;
                                *)
                                        if [ ! "${vHost}" = "" ]
                                        then

                                                # echo "${vHost}"
                          ping -i 1 -c 1 -w 1 "${vHost}" >/dev/null 2>&1
                                                vError=$?
                                                if [ ${vError} -eq 1 ]
                                                then
                                    print "Please delete host: ${vHost}"
                                                fi

                                        fi
                                        ;;
                        esac
                done
        done
```

Robert Kaiser, Systems Analyst
Bayerischer Rundfunk (Germany)

Business-driven capacity planning for AIX: concepts and principles

"How can my revenue stay flat in 2004 when I have to increase my spending on AIX servers by 35%? The business plans to double its sales of widgets in the next six months; does that mean I have to double my server resources or can I get by with what I have?" These are common questions posed over and over again by businesses to their capacity planners and their management.

This set of articles describes an approach that addresses these types of question. The first part, *concepts and principles*, focuses on the essential building blocks for aligning capacity planning with the business. The second part, *techniques and approaches* (next issue), describes the 'how-tos' for this approach. The final article, *applying the techniques*, walks through the application of the techniques via a case study.

KEY STEPS

There are eight key steps in business-driven capacity planning. The critical aspect of this process is to establish a good working relationship with the business application owner and the person in the Line of Business (LOB) who knows the operational metrics and does the LOB business planning. Establishing this will result in briefer planning meetings and more timely notification of business changes that will impact on the server capacity plan.

The eight steps are:

1 Identify applications of interest.

2 Identify the key business metrics used in business planning and operational measurements.

3 Identify a candidate set of business metrics that might correlate with your AIX application or 'transactions'.

4 Determine a level of correlation and develop a regression curve for projections.

5 Identify key business events and activities that may affect application volumes or changes in mix.

6 Understand current usage characteristics and patterns over time.

7 Translate business forecasts and business events into volume changes over time.

8 Identify the configuration to meet service targets at each major projected volume change.

KEY CONCEPTS

Proceeding through those eight steps requires some knowledge of the business-driven concepts and principles. We'll cover those before going through the steps and techniques.

A business driver is a business metric such as number of cheques processed, number of calls received, number of claims processed, number of incidents reported, number of claims rejected. A business driver is also one that is measured by the business on a daily and preferably hourly basis – although it is possible to do gross correlations with monthly business metrics.

A business application or business transaction is the most granular level for business-driven forecasting. Within AIX, this would be a grouping of processes or threads into a business application or business transaction. This grouping can be done either by using Application Resource Measurement (ARM) to implement the code, or by creating APPLIDs to associate process ids with applications (by APPLID) using products from IBM, BMC, or HP. The process mapping allows for CPU time to be associated with each APPLID (business transaction or application) and can be correlated with a business metric.

Business driver correlation is the method of quantifying the relationship between a business driver and the corresponding IT metric (typically CPU time per application or transaction, but possibly storage – it might be GB-months). It begins with determining whether a correlation exists between a candidate business metric and a business transaction or application. If the correlation coefficient between those two metrics is greater than 0.65, then regression work should be done to determine the equation that provides the best fit for those two variables.

FORECASTING TECHNIQUES

Figure 1 illustrates two techniques:

1 Business driver

2 Business transaction.

The business driver technique focuses on correlating one business metric with one business application (a set of business transactions). The business transaction technique focuses on correlating a specific business metric for each individual business transaction. In this case, there will always be a one-to-one correlation. In the business driver case, this may not be possible. However, this is the approach that should be used first.

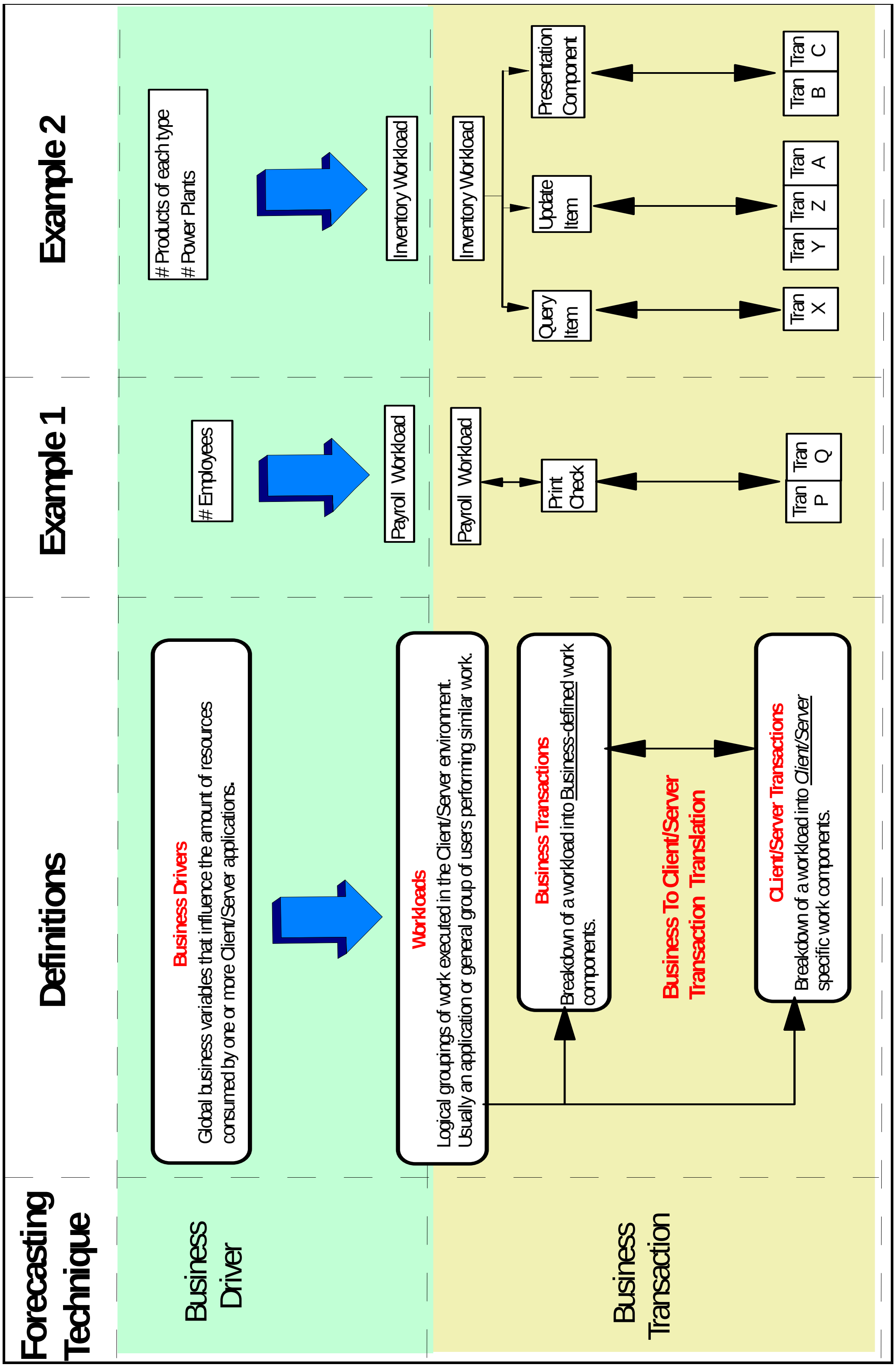
BUSINESS DRIVER TECHNIQUE EXAMPLE

For simplicity, we are going to use an example that doesn’t involve statistics, but is really just an intuitive approach. In Figure 2 we see two candidate business drivers for workload XYZ:

1 Number of users

2 Number of projects.

These two business drivers are candidates for correlating with AIX CPU seconds for the workload XYZ. Looking at the data for the first case shows an obvious correlation. Looking at the second driver, number of projects, we see that it has some



Forecasting Technique

Definitions

Example 1

Example 2

Business Driver

Business Transaction

Business Drivers
Global business variables that influence the amount of resources consumed by one or more Client/Server applications.

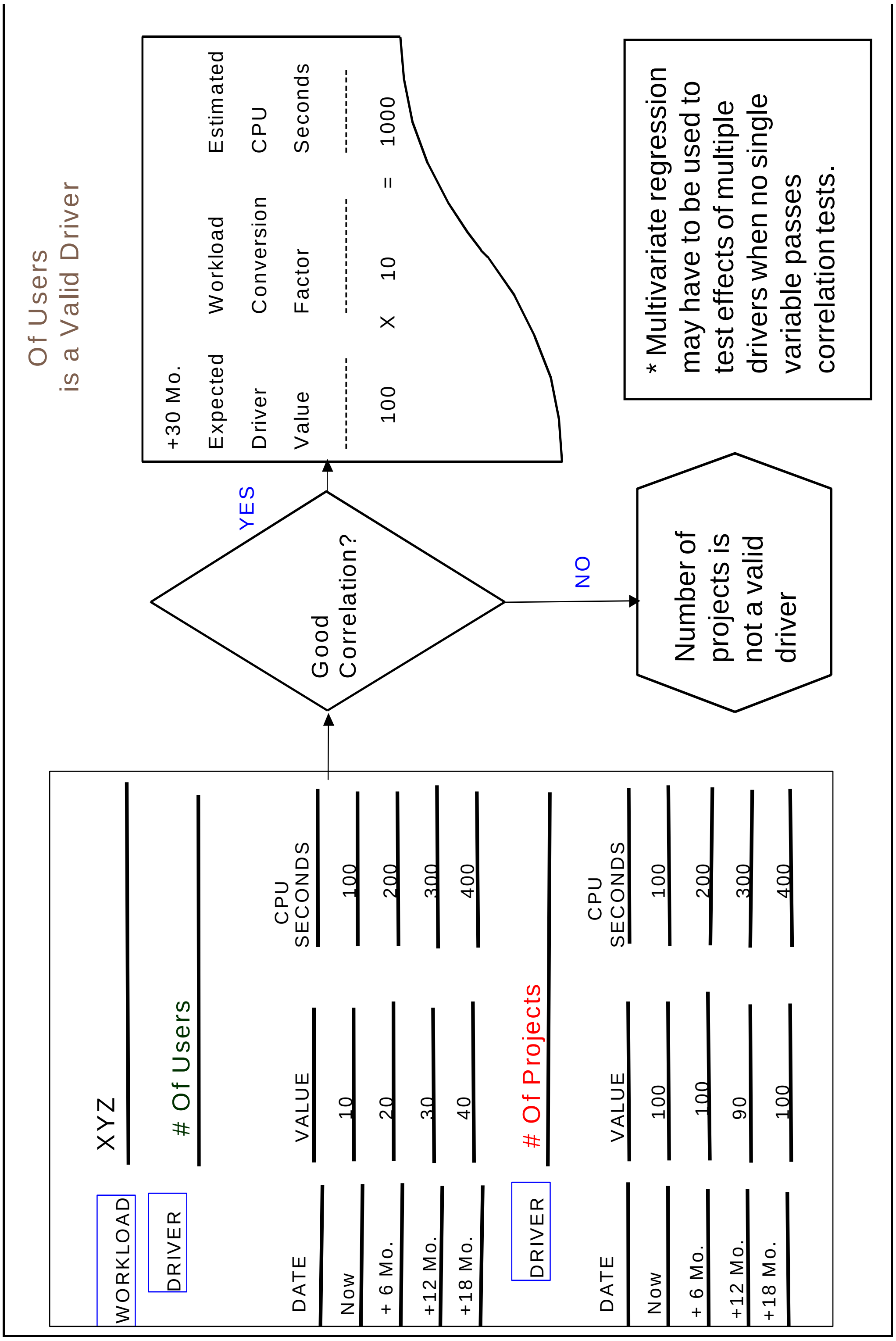
Workloads
Logical groupings of work executed in the Client/Server environment. Usually an application or general group of users performing similar work.

Business Transactions
Breakdown of a workload into Business-defined work components.

Business To Client/Server Transaction Translation

CLient/Server Transactions
Breakdown of a workload into *Client/Server* specific work components.

Figure 1: ***Business driver and transaction techniques***



WORKLOAD	XYZ	
DRIVER	# Of Users	
DATE	VALUE	CPU SECONDS
Now	10	100
+ 6 Mo.	20	200
+12 Mo.	30	300
+18 Mo.	40	400

DRIVER	# Of Projects	
DATE	VALUE	CPU SECONDS
Now	100	100
+ 6 Mo.	100	200
+12 Mo.	90	300
+18 Mo.	100	400

+30 Mo. Expected Driver Value		Workload Conversion Factor		Estimated CPU Seconds
100	X	10	=	1000

Of Users
is a Valid Driver
Good Correlation?
YES
NO
Number of projects is not a valid driver
* Multivariate regression may have to be used to test effects of multiple drivers when no single variable passes correlation tests.

Figure 2: Candidate business drivers

anomalous behavior and doesn't correlate with CPU seconds. So, number of users is the business driver and a good business predictor of CPU growth to use. The Workload Conversion Factor (WCF) would normally be derived, but in this case it can clearly be seen to be a factor of 10.

SUMMARY

In this article, we've provided a high-level perspective of business-driven capacity planning. In the next two articles, we'll describe more details on how to make this a reality in doing AIX capacity forecasts.

Russ Egeland
Managing Consultant
IBM Global Services (USA)

System configuration lister

The configuration of any AIX system is (still, in the 21st century) a complex process of definitions, with many complex and interrelated details. The following script is an attempt to collect and present this information in simple human-readable forms – HTML. The script can be used as a stand-alone utility or as part of a Web-based system information repository. The script is called sysdoc.

SYSDOC SCRIPT

```
#
# Start of Sysdoc Script
#!/usr/bin/ksh
# Print system documentation for RS/6000 and pSeries running AIX 5.X
function printtitle
{
echo "<P><TITLE>"
echo $*
echo "</TITLE>"
}

function printhead
{
echo "<P><H1>"
echo $*
echo "</H1>"
}

function printhead2
{
echo "<P><H2>"
echo "<A NAME=\"$2\">$1</A>"
echo "</H2>"
}

function printhead3
{
echo "<P><H3>"
echo "<A NAME=\"$2\">$1</A>"
echo "</H3>"
echo "<A HREF=\"#Ø\">Back</A>"
}

function printhead3_no_backlink
{
echo "<P><H3>"
echo "<A NAME=\"$2\">$1</A>"
echo "</H3>"
}

function printtext
{
echo "<P><PRE>"
echo $* | awk '{print "<BR>" $Ø}'
echo "</PRE>"
}

function printcom
{
if [[ "$VERBOSE" = "TRUE" ]]
then
  echo "<B>"
  echo  "The following output has been produced by command: "
  echo "<I>"
  echo $*
  echo "</B></I>"
fi
  echo "<P><PRE>"
  $* | awk '{print "<BR>" $0}'
  echo "</PRE>"
}

function printcom_and_text
{
      printhead3_no_backlink "$2"
      printcom  $1
}

function printcom2
{
if [[ "$VERBOSE" = "TRUE" ]]
then
  echo "<B>"
  echo  "The following output has been produced by command: "
  echo "<I>"
  echo $1 '|' $2
  echo "</B></I>"
fi
  echo "<P><PRE>"
  $1 | $2 | awk '{print "<BR>" $0}'
  echo "</PRE>"
}

function printcom_trim
{
if [[ "$VERBOSE" = "TRUE" ]]
then
  echo "<B>"
  echo  "The following output has been produced by command: "
  echo "<I>"
  echo $*
  echo "</B></I>"
fi
  echo "<P><PRE>"
  $* | sed '/^#.*/d' | tr ':' '\011' | awk '{print "<BR>" $0}'
  echo "</PRE>"
}

function printfile
{
if [[ "$VERBOSE" = "TRUE" ]]
then
  echo "<B>"
  echo  "The following output has been produced by command: "
  echo "<I>"
  echo "cat $1 | sed '/^[#:\*].*/d'"
  echo "</B></I>"
fi
```

```
if [[ -f $1 ]]
then
  printhead3_no_backlink $1
  echo "<P><PRE>"
      egrep -v '^#|^[       ]*$' $1 # remove comment and empty lines
  echo "</PRE>"
else
  printhead3_no_backlink $1
  printhead3_no_backlink "not found on this system"
fi
}

function STD
{
        exec 3>&1
        ($* 2>&1 1>&3) | sed -e 's/^/<FONT
 COLOR=\"RED\"><B>ERROR: /' -e
's/$/<\/B><\/FONT>/' >&1
}

function printAnchor
{
      echo "<TR><TD>"
      echo "<A HREF=\"#$3\">$1</A>"
      echo "</TD><TD>"
      echo "<A HREF=\"#$3\">$2</A>"
      echo "</TD></TR>"
}

XP_DISKS()
{
  printcom echo "Creating XP Disks Layout ...."
  printcom xpinfo
}
#-----------------------------------------
# EMC Prodcedure
#-----------------------------------------
EMC_DISKS()
{
      printcom echo "Creating EMC Disks Layout ...."
      if [ -x /usr/symcli/bin/syminq ]
      then
            printcom /usr/symcli/bin/syminq -sym
      else
            echo "<FONT COLOR=\"RED\"><B>WARNING: </B></FONT><BR>"
            echo "<FONT COLOR=\"RED\"><B>WARNING: There is no
executable /usr/symcli/bin/syminq</B></FONT><BR>"

            echo "<FONT COLOR=\"RED\"><B>WARNING: to output the
information about the SYMMETRIX storage</B></FONT><BR>"
```

```
            echo "<FONT COLOR=\"RED\"><B>WARNING: </B></FONT><BR>"
      fi
}
#-----------------------------------------
# DGC Prodcedure - CLARiiON CX600 Array
#-----------------------------------------
#Note that this procedure merely identifies the disks by name,
#  but IS NOT able
#to determine the number of links to the storage arrary.
#Multiple links will multiply the total capacity by the number of links
!

DGC_DISKS()
{
      printcom echo "Creating CLARiiON Disks Layout ...."
      if [ -x /usr/symcli/bin/syminq ]
      then
            printcom /usr/symcli/bin/syminq -powerpath
      else
            echo "<FONT COLOR=\"RED\"><B>WARNING: </B></FONT><BR>"
            echo "<FONT COLOR=\"RED\"><B>WARNING: There is no
executable /usr/symcli/bin/syminq</B></FONT><BR>"

            echo "<FONT COLOR=\"RED\"><B>WARNING: to output the
information about the CLARiiON storage</B></FONT><BR>"

            echo "<FONT COLOR=\"RED\"><B>WARNING: </B></FONT><BR>"
      fi
}

AIX_STANDARD_DISKS()
{
   printcom echo "Creating Standard Disks Layout ...."
   printcom lsdev -Cc disk | grep LVD
}
#-----------------------------------------
# AIX FibreCard RAID Prodcedure
#-----------------------------------------
AIX_FC_RAID_DISKS()
{
   echo "Creating FibreCard RAID Disks Layout"
}
#-----------------------------------------
# AIX SSA disks
#-----------------------------------------
AIX_SSA_DISKS()
{
## echo "Creating SSA Disks Layout ...."
for i in 'ssadisk -a ssa0 -P'
```

```
do
   printcom lsdev -C -l $i
done
}

POWER_PATH()
{
      printcom powermt display
      printcom powermt display dev=all
}
HDLM_PATH()
{
      BIN=/usr/DynamicLinkManager/bin/dlnkmgr
      printcom $BIN view -sys
      printcom $BIN view -path
}

MPIO_PATH()
{
        printtext "MPIO Paths"
      printcom2 "lspath" "sort +1"
      for disk in $(lspath|awk '{print $2}'|sort -u)
      do
             printcom_and_text "lsattr -El $disk" "Attributes of $disk"
            printcom_and_text "lspath -l  $disk -r parent"
"Parent(s) of $disk"
      done
}

DATA_PATH()
{
  printcom datapath query adapter
  printcom datapath query device
  printcom lsvpcfg
}

AIX_Layout()
{
LOGIOSCAN=/tmp/Ioscan_'hostname'
XPLOG=/tmp/xp.log
XPLOG2=/tmp/xp2.log
XPLOG3=/tmp/xp3.log
VA7100LOG=/tmp/va7100.log
STANDLOG=/tmp/stand.log
AUTOLOG=/tmp/auto.log
AUTOLOG2=/tmp/auto2.log
SUMMARY1LOG=/tmp/summary1.log
EMCLOG=/tmp/emc.log
EMCLOG2=/tmp/emc2.log
EMCLOG3=/tmp/emc3.log
```

```
A5000=/tmp/a5000.log
CCISSLOG=/tmp/cciss.log
XPFLAG=OPEN
EMCFLAG=EMC
DGCFLAG=DGC

   BINDIR="/usr/local/ADMIN/INFOUX/bin/"
   lsdev -Cc disk > $LOGIOSCAN
   XPINFO='whence xpinfo'
   [ $? -eq 0 ] && $XPINFO -i >> $LOGIOSCAN

   AIX_STANDARD_DISKS
   INQ=$BINDIR/IBM_inq
   [ $? -eq 0 ] && $INQ 2>/dev/null >> $LOGIOSCAN

   [ -n "$(cat $LOGIOSCAN | grep $XPFLAG)" ] && XP_DISKS
   [ -n "$(cat $LOGIOSCAN | grep $EMCFLAG)" ] && EMC_DISKS
   [ -n "$(cat $LOGIOSCAN | grep $DGCFLAG)" ] && DGC_DISKS

   for i in $(lsdev -CS1 -cpdisk -sssar -F name)
   do
      echo "$i: "$(ssaxlate -l $i) > /tmp/ssa
   done
   [ -s /tmp/ssa ] && AIX_SSA_DISKS
}

MultiPathing()
{
   POWERFLAG=PowerPath
   HDLMFLAG=AutoPath
   MPIOFLAG=MPIO

   [ -n "$(lslpp -l | grep $POWERFLAG)" ] && POWER_PATH
   [ -n "$(lslpp -l | grep $HDLMFLAG)" ] && HDLM_PATH
   [ -n "$(lsdev -Cc disk | grep $MPIOFLAG)" ] && MPIO_PATH

   if [ -f /usr/sbin/lsvpcfg ]
      then
         [ -n "$(/usr/sbin/lsvpcfg|wc -c)" ] && DATA_PATH
   fi
}

GLOBAL_INF()
{

# System information
msize=0
for i in 'lscfg | grep -i mem | awk '{print $2}''
do
        ((msize=msize + 'lsattr -El mem0 | awk '/^size/ {print $2}''))
```

```
done
procstr='lsdev -Cc processor'

procspeed='lsattr -El ${procstr%% *} -a frequency -F value'
cputype='getsystype -y'
kerntype='getsystype -K'
lparinfo='uname -L'

# Pre-5.2, 'uname -L' returns a leading "AIX"
# Check for a leading AIX and remove it
if [[ $lparinfo = AIX* ]]
then
lparinfo=${lparinfo#* }
fi

# Ensure that procspeed is not null,
# since the conversion to MHz below will
# cause the script to exit with an error if null.
if [[ ! -z $procspeed ]]
then
    # Convert procspeed to MHz, and round up or down
    if (( ($procspeed%1000000) >= 500000 ))
    then
    (( procspeedMHz=($procspeed/1000000) + 1 ))
    else
    (( procspeedMHz=($procspeed/1000000) ))
    fi
fi
model='lsattr -El sys0 -a modelname -F value'
# Get Serial number
serial='lscfg -vpl sysplanar0 |/usr/bin/grep -p "System:" |grep
"Machine/Cabinet"`
if [[ $? -eq 0 ]]; then
    serial=${serial##*.}
else
    serial='lscfg -vpl sysplanar0 |/usr/bin/grep -p "System VPD:" |grep
"Machine/Cabinet"`
    if [[ $? -eq 0 ]]; then
        serial=${serial##*.}
    else
        noserial=1
    fi
fi

gsize='lsattr -El mem0 | awk '/^goodsize/ {print $2}''
proc_no='lsdev -Cc processor | head -1 | cut -d' ' -f1'
proctype='lsattr -El $proc_no | awk '/^type/ {print $2}''
numproc='lsdev -Cc processor | grep Available | wc -l'
numproc=${numproc##* }
fwversion='lsattr -El sys0 -a fwversion -F value'
```

```
conslogin='lsattr -El sys0 -a conslogin -F value'
autorestart='lsattr -El sys0 -a autorestart -F value'
fullcore='lsattr -El sys0 -a fullcore -F value'

/usr/bin/dspmsg -s 1 cmdsolsysmgt.cat 1 "System Model:"
echo " ${model}"

#Print out serial if found, otherwise "Not available"
if [[ -z $noserial ]] then
    /usr/bin/dspmsg -s 1 cmdsolsysmgt.cat 52 "Machine Serial Number:"
    echo " ${serial}"
else
    /usr/bin/dspmsg -s 1 cmdsolsysmgt.cat 52 "Machine Serial Number:"
    print " \c"
    /usr/bin/dspmsg -s 1 cmdsolsysmgt.cat 53 "Not Available"
fi

/usr/bin/dspmsg -s 1 cmdsolsysmgt.cat 5 "Processor Type:"
echo " ${proctype}"

/usr/bin/dspmsg -s 1 cmdsolsysmgt.cat 4 "Number Of Processors:"
echo " ${numproc}"

/usr/bin/dspmsg -s 1 cmdsolsysmgt.cat 48 "Processor Clock Speed:"
echo " ${procspeedMHz} MHz"

/usr/bin/dspmsg -s 1 cmdsolsysmgt.cat 49 "CPU Type:"
echo " ${cputype}-bit"

/usr/bin/dspmsg -s 1 cmdsolsysmgt.cat 50 "Kernel Type:"
echo " ${kerntype}-bit"

/usr/bin/dspmsg -s 1 cmdsolsysmgt.cat 51 "LPAR Info:"
echo " ${lparinfo}"

/usr/bin/dspmsg -s 1 cmdsolsysmgt.cat 2 "Memory Size:"
echo " ${msize} MB"

/usr/bin/dspmsg -s 1 cmdsolsysmgt.cat 3 "Good Memory Size:"
echo " ${gsize} MB"

/usr/bin/dspmsg -s 1 cmdsolsysmgt.cat 6 "Firmware Version:"
echo " ${fwversion}"

/usr/bin/dspmsg -s 1 cmdsolsysmgt.cat 7 "Console Login:"
echo " ${conslogin}"

/usr/bin/dspmsg -s 1 cmdsolsysmgt.cat 8 "Auto Restart:"
echo " ${autorestart}"
```

```
/usr/bin/dspmsg -s 1 cmdsolsysmgt.cat 9 "Full Core:"
echo " ${fullcore}"

/usr/bin/dspmsg -s 1 cmdsolsysmgt.cat 777 "Multi-Processor Capability:"
   (( $(bootinfo -z) == 0 )) && echo No || echo Yes
/usr/bin/dspmsg -s 1 cmdsolsysmgt.cat 777 "Boot List::"
   printcom "bootlist -m normal -o"
/usr/bin/dspmsg -s 1 cmdsolsysmgt.cat 777 "Booted From::"
   printcom "bootinfo -b"
   echo "Display System Rset Contents"
   printcom "lsrset -av"
   echo "Uptime, What"
   printcom "w"
   echo "Display Summary of Statistics since boot"
   printcom "vmstat"
   echo "Display Fork statistics"
   printcom "vmstat -f"
   echo  "Displays the number of Interrupts"
   printcom "vmstat -i"
   echo  "List Sum of paging events"
   printcom "vmstat -s"
   echo "Report CPU and I/O statistics"
   printcom "iostat"
   echo  "List Kernel Lock Tracing current Status"
   printcom "locktrace -l"
}
#
# Print 2 lines of mandatory output
#
echo '<HTML>'
echo
#
# Start of the main program
####################################################################
HOSTNAME=$(hostname)
DATE=$(date)

IT="<I>"
E_IT="</I>"
PDISKS='lspv | grep active | awk '{print $1}''
VGS='lsvg -o'  # List VG infor for available VG's only !
#
# Options Setting
#
cc='echo $QUERY_STRING|grep VERBOSE|wc -c'
if [ $cc -gt 0 ]
then
 VERBOSE="TRUE"
fi
```

```
cc='echo $QUERY_STRING|grep SOFTWARE|wc -c'
if [ $cc -gt Ø ]
then
 SOFTWARE="TRUE"
fi

cc='echo $QUERY_STRING|grep HARDWARE|wc -c'
if [ $cc -gt Ø ]
then
 HARDWARE="TRUE"
fi

cc='echo $QUERY_STRING|grep STORAGE|wc -c'
if [ $cc -gt Ø ]
then
 STORAGE="TRUE"
fi

cc='echo $QUERY_STRING|grep USERS|wc -c'
if [ $cc -gt Ø ]
then
 USERS="TRUE"
fi

cc='echo $QUERY_STRING|grep COMMS|wc -c'
if [ $cc -gt Ø ]
then
 COMMS="TRUE"
fi

cc='echo $QUERY_STRING|grep SYSENV|wc -c'
if [ $cc -gt Ø ]
then
 SYSENV="TRUE"
fi

cc='echo $QUERY_STRING|grep LUM|wc -c'
if [ $cc -gt Ø ]
then
 LUM="TRUE"
fi

cc='echo $QUERY_STRING|grep MISCFILES|wc -c'
if [ $cc -gt Ø ]
then
 MISCFILES="TRUE"
fi

cc='echo $QUERY_STRING|grep CRONFILES|wc -c'
if [ $cc -gt Ø ]
```

```
then
 CRONFILES="TRUE"
fi

SOFTWARE="TRUE"
HARDWARE="TRUE"
STORAGE="TRUE"
SYSENV="TRUE"
USERS="TRUE"
COMMS="TRUE"
LUM="TRUE"
MISCFILES="TRUE"
CRONFILES="TRUE"
if [[  "$SOFTWARE" = "TRUE" || "$HARDWARE" = "TRUE" ||  "$STORAGE" =
"TRUE" ||  "$USERS" = "TRUE" || "$COMMS" = "TRUE" ||  "$SYSENV" = "TRUE"
|| "$DATAPATHFLAG" = "TRUE" || "$LUM" = "TRUE" || "$MISCFILES" = "TRUE"
|| "$CRONFILES" = "TRUE" ]]
then
  printtitle Configuration information for host $HOSTNAME on $DATE
  printhead Configuration information for host $IT$HOSTNAME$E_IT on
$IT$DATE$E_IT
else
  printhead2 "No Sections Selected" 0
fi
##################################################
# Here comes the contents table
##################################################
      printhead2 "Contents:" 0
      echo "<TABLE>"
      if [[ "$SOFTWARE" = "TRUE" ]]
      then
            printAnchor Software "<BR>" 1
            printAnchor "<BR>" "Level of AIX Operating System" 1.1
            printAnchor "<BR>" "Installed Software" 1.2
              printAnchor "<BR>" "Extended Installed Software
Information" 1.3
      fi
      if [[ "$HARDWARE" = "TRUE" ]]
      then
            printAnchor Hardware "<BR>" 2
            printAnchor "<BR>" "Global Parameters" 2.01
            printAnchor "<BR>" "System Parameters" 2.1
            printAnchor "<BR>" "Installed Devices" 2.2
            printAnchor "<BR>" "Extended Hardware Configuration" 2.3
      fi
      if [[ "$STORAGE" = "TRUE" ]]
      then
            printAnchor Storage "<BR>" 4
            printAnchor "<BR>" "Storage information" 4.01
            printAnchor "<BR>" "MultiPathing information" 4.02
            printAnchor "<BR>" "Installed Physical Disks" 4.1
            printAnchor "<BR>" "Volume Groups" 4.2
            printAnchor "<BR>" "Online Volume Groups" 4.3
            printAnchor "<BR>" "Volume Groups Characteristics" 4.4
            printAnchor "<BR>" "Physical Disks Distribution per
Volume Group" 4.5
            printAnchor "<BR>" "Logical Volumes Distribution per
Volume Group" 4.6
            printAnchor "<BR>" "Logical Volumes Distribution per
Physical Disk" 4.7
            printAnchor "<BR>" "Logical Volumes Characteristics" 4.8
           printAnchor "<BR>" "Paging Space Layout and Utilization" 4.9
            printAnchor "<BR>" "File Systems" 4.10
            printAnchor "<BR>" "Mounted File Systems" 4.11
      fi
      if [[ "$USERS" = "TRUE" ]]
      then
            printAnchor "Users Information" "<BR>" 5
            printAnchor "<BR>" Users 5.1
            printAnchor "<BR>" Groups 5.2
            printAnchor "<BR>" Roles 5.3
      fi
      if [[ "$COMMS" = "TRUE" ]]
      then
            printAnchor "Communications" "<BR>" 6
            printAnchor "<BR>" Hostname 6.1
            printAnchor "<BR>" "Network Status" 6.2
            printAnchor "<BR>" "Arp Table" 6.3
            printAnchor "<BR>" "Routing Table" 6.4
            printAnchor "<BR>" "Network Interfaces" 6.5
            printAnchor "<BR>" "Name Resolution /etc/hosts" 6.6
            printAnchor "<BR>" "Name Resolution /etc/resolv.conf" 6.7
            printAnchor "<BR>" "Client Network Services
/etc/services" 6.8
            printAnchor "<BR>" "Protocols /etc/protocols" 6.9
            printAnchor "<BR>" "Syslog Configuration
/etc/syslog.conf" 6.10
            printAnchor "<BR>" "Remote Host Access Control
/etc/hosts.equiv"  6.11
            printAnchor "<BR>" "NFS and NIS Info " 6.11
      fi
      if [[ "$SYSENV" = "TRUE" ]]
      then
            printAnchor "System Environments" "<BR>" 7
            printAnchor "<BR>" "/etc/inittab" 7.1
            printAnchor "<BR>" "Subsystems" 7.2
            printAnchor "<BR>" TimeZone 7.3
      fi
      if [[ "$LUM" = "TRUE" ]]
      then
```

```
            printAnchor "License Use Manager" "<BR>" 8
      fi
      if [[ "$MISCFILES" = "TRUE" ]]
      then
            printAnchor "Miscellanious Files" "<BR>" 9
      fi
      if [[ "$CRONFILES" = "TRUE" ]]
      then
            printAnchor "Cron Files" "<BR>" 10
            printAnchor "<BR>" "Cron Allow and Deny Files"  10.1
            printAnchor "<BR>" "Users Cron Files"  10.2
      fi
      echo "</TABLE>"
###################################################
# Until here comes the contents table
###################################################
  if [[ -f /usr/local/ADMIN/etc/Machine.profile ]]
  then
    printhead3 Machine Profile file
    printfile /usr/local/ADMIN/etc/Machine.profile
  fi

if [[ "$SOFTWARE" = "TRUE" ]]
then
  printhead2 Software 1

  printhead3 "Level of AIX Operating System" 1.1
  printcom oslevel  -r
  printhead3 "Installed Software" 1.2
  printcom STD lslpp -l
  printhead3 "Extended Installed Software Information" 1.2
  printcom STD lslpp -La

fi

if [[ "$HARDWARE" = "TRUE" ]]
then
  printhead2 "Hardware" 2
  printhead3 "Global Parameters" 2.01
  printcom GLOBAL_INF
  printhead3 "System Parameters" 2.1
  printcom STD lsattr -E -H -l sys0
  printhead3 "Installed Devices" 2.2
  printcom STD "lsdev -C"
  printhead3 "Extended Hardware Configuration" 2.3
  printcom STD "lscfg -pv"

fi

if [[ "$STORAGE" = "TRUE" ]]
```

```
then
  printhead2 Storage 4
  printhead3 "Storage information" 4.Ø1
  AIX_Layout
  printhead3 "MultiPathing information" 4.Ø2
  MultiPathing
  printhead3 "Installed Physical Disks" 4.1
  printcom  STD lspv

  printhead3 "Volume Groups" 4.2
  printcom STD lsvg
  printhead3 "Online Volume Groups" 4.3
  printcom STD lsvg -o
  printhead3 "Volume Groups Characteristics" 4.4
  for i in $VGS
  do
   printcom STD lsvg $i
  done

  printhead3 "Physical Disks Distribution per Volume Group" 4.5
  for i in $VGS
  do
   printcom STD lsvg -p $i
  done

  printhead3 "Logical Volumes Distribution per Volume Group" 4.6
  for i in $VGS
  do
   printcom STD lsvg -l $i
  done
  printhead3 "Logical Volumes Distribution per Physical Disk" 4.7
  for i in $PDISKS
  do
   printcom STD lspv -l $i
  done
  printhead3 "Logical Volumes Characteristics" 4.8
  for j in $VGS
  do
   LVS='lsvg -l $j|grep '/'|awk '{print $1}''
   for i in $LVS
   do
    printcom STD lslv $i
   done
  done
  printhead3 "Paging Space Layout and Utilization" 4.9
  for PS in 'lsps -a -l'
  do
      printcom echo $PS:
      printcom STD lsps  $PS
  done
```

```
#  printcom lsps -a

  printhead3 "File Systems" 4.10
  printcom STD lsfs -a -q

  printhead3 "Mounted File Systems" 4.11
  printcom mount

fi

if [[ "$USERS" = "TRUE" ]]
then
  printhead2 "Users Information" 5
  printhead3 Users 5.1
  printtext "Name    Id   Group(s) Home Directory   Shell"
  printcom_trim lsuser -c  ALL
  printhead3 Groups 5.2
  printtext "Name  Id Admin Members"
  printcom_trim lsgroup -c  ALL
  printhead3 Roles 5.3
  printcom_trim lsrole -c  ALL
fi

if [[ "$COMMS" = "TRUE" ]]
then
  printhead2 Communications 6
  printhead3 Hostname 6.1
  printcom hostname
  printhead3 "Network Status" 6.2

   printcom_and_text "netstat -in" "List of all IP addresses"
   printcom_and_text "ifconfig -a" "Display information about all
network interfaces"
   printcom_and_text "no -a" "Display current network attributes in the
kernel"

   printcom_and_text "netstat -nr" "List of all routing table entries by
IP-address"

   printcom_and_text "netstat -an" "Show the state of all sockets"
   printcom_and_text "netstat -An" "Show the address of any PCB
associated with the sockets"

   printcom_and_text "netstat -s" "Show statistics for each protocol"
   printcom_and_text "netstat -sr" "Show the routing statistics"
   printcom_and_text "netstat -v" "Show statistics for CDLI-based
communications adapters"
   printcom_and_text "netstat -m" "Show statistics recorded by memory
management routines"
```

```
  printhead3 "Arp Table" 6.3
  printcom arp -a
  printhead3 "Routing Table" 6.4
  printcom netstat -r
  printhead3 "Network Interfaces" 6.5
  printcom lsdev -C -c if
  printhead3 "Name Resolution file" 6.6
  printfile /etc/hosts
  printhead3 "Name Resolution file" 6.7
  printfile /etc/resolv.conf
  printhead3 "Client Network Services file" 6.8
  printfile /etc/services
  printhead3 "Protocols file" 6.9
  printfile /etc/protocols
  printhead3 "Syslog Configuration file" 6.10
  printfile  /etc/syslog.conf
  printhead3 "Remote Host Access Control file"  6.11
  printfile /etc/hosts.equiv
  printhead3 "NFS and NIS Info" 6.12
  printcom_and_text "lsfs -v nfs" "NFS filesystems mounted from remote
hosts"
  printhead3_no_backlink "Local Directories Exported by NFS"
  printfile /etc/exports
  printcom_and_text "nfso -a" "List Network File System (NFS) network
variables"
  printcom_and_text "rpcinfo" "Display a List of Registered RPC
Programs"
  printcom_and_text "rpcinfo -m" "rpcinfo -m"
  printcom_and_text "rpcinfo -s" "rpcinfo -s "
  printcom_and_text "lsnamsv -C" "DNS Resolver Configuration: lsnamsv
-C"
  printcom_and_text "namerslv -s" "Display all Name Server Entries"
  printcom_and_text "domainname" "NIS Domain Name"
  printcom_and_text "ypwhich" "NIS Server currently used"
  printcom_and_text "lsclient -l" "NIS Client Configuration"
fi

if [[ "$SYSENV" = "TRUE" ]]
then
  printhead2 "System Environments" 7

  printhead3 "Inittab file" 7.1
  printfile /etc/inittab

  printhead3 Subsystems 7.2
  printcom lssrc -a

  printhead3 TimeZone 7.3
  printtext $TZ
fi
```

```
########################################################################
#       LUM License Configuration
########################################################################
if [ "$LUM" = "TRUE" ]
then
    printhead2 "License Use Manager" 8
    printhead3 "" 8

   for FILE in  /var/ifor/nodelock /var/ifor/i4ls.ini /var/ifor/i4ls.rc
/etc/ncs/glb_site.txt /etc/ncs/glb_obj.txt
   do
      printfile "${FILE}"
   done
   /usr/opt/ifor/bin/i4cfg -list | grep -q 'active'
   rc=$?
   if (( $rc == 0 ))
   then
      printcom_and_text "/usr/opt/ifor/bin/i4blt -ll -n $(uname -n)"
"Installed Floating Licenses"
      printcom_and_text "/usr/opt/ifor/bin/i4blt -s -n $(uname -n)"
"Status of Floating Licenses"
   fi

   printcom_and_text "inulag -lc" "License Agreements Manager"
   printcom_and_text "lslicense" "Display fixed and floating OS Users
Licenses"

fi
if [ "$MISCFILES" = "TRUE" ]
then
    printhead2 "Miscellanious Files" 9
    printhead3 "" 9

   files(){
echo      /etc/aliases
echo      /etc/binld.cnf
echo      /etc/bootptab
echo      /etc/dhcpcd.ini
echo      /etc/dhcprd.cnf
echo      /etc/dhcpsd.cnf
echo      /etc/dlpi.conf
echo      /etc/environment
echo      /etc/ftpusers
echo      /etc/gated.conf
echo      /etc/hostmibd.conf
echo      /etc/hosts.lpd
echo      /etc/inetd.conf
echo      /etc/mib.defs
echo      /etc/mrouted.conf
echo      /etc/netgroup
echo      /etc/netsvc.conf
echo      /etc/ntp.conf
echo      /etc/oratab
echo      /etc/policyd.conf
echo      /etc/pse.conf
echo      /etc/pse_tune.conf
echo      /etc/pxed.cnf
echo      /etc/qconfig
echo      /etc/filesystems
echo      /etc/rc
echo      /etc/rc.adtranz
echo      /etc/rc.bsdnet
echo      /etc/rc.licstart
echo      /etc/rc.net
echo      /etc/rc.net.serial
echo      /etc/rc.oracle
echo      /etc/rc.qos
echo      /etc/rc.shutdown
echo      /etc/rc.tcpip
echo      /etc/rsvpd.conf
echo      /etc/sendmail.cf
echo      /etc/slip.hosts
echo      /etc/snmpd.conf
echo      /etc/snmpd.peers
echo      /etc/telnet.conf
echo      /etc/xtiso.conf
echo      /opt/ls3/ls3.sh
echo      /usr/tivoli/tsm/client/ba/bin/rc.dsmsched
echo      /usr/tivoli/tsm/server/bin/rc.adsmserv
find /etc/rc.d -type f -print
}

   for FILE in $(files)
   do
      printfile "${FILE}"
   done

fi
if [ "$CRONFILES" = "TRUE" ]
then
    printhead2 "Cron Files" 1Ø
    printhead3 "Cron Allow and Deny Files" 1Ø.1
    printfile /var/adm/cron/cron.allow
    printfile /var/adm/cron/cron.deny

   files(){
      find /var/spool/cron/crontabs -type f -print
      }

   printhead3 "Users Crontab Files" 1Ø.2
   for FILE in $(files)
   do
      printfile "${FILE}"
   done
   printcom_and_text "at -l" "AT Jobs"

fi
echo '</HTML>'
# End Of Sysdoc Script
```

Alex Polak
System Engineer
APS (Israel)

AIX news

IBM Printing Systems has announced enhanced functionality in Version 4.1 of its Infoprint Manager solution on AIX to natively drive Metacode datastreams. This provides commercial and in-plant print customers with greater control and output management for multi-vendor environments.

There is now support for Line Conditioned Data Stream (LCDS) and Metacode. With the LCDS and Metacode feature, Infoprint Manager for AIX can support the datastreams used by Xerox EPS printers. Alternatively, Infoprint Manager for AIX is supported by Advanced Function Printing (AFP) architecture to combine datastream transforms and output management capabilities into a single solution.

For further information contact your local IBM representative.
URL: www.ibm.com/press/PressServletForm.wss?TemplateName=ShowPressReleaseTemplate&SelectString=t1.docunid=7264.

* * *

VERITAS Software has announced that its storage management offerings now include new features for AIX. These AIX-based additions include Version 4.0 of its Storage Foundation, Storage Foundation for Oracle RAC, Storage Foundation for Databases (DB2 and Oracle), Storage Foundation Cluster File System, Cluster Server, and Volume Replicator software.

For further information contact:
VERITAS Software, 350 Ellis Street, Mountain View, CA, 94043, USA.
Tel: (650) 527 8000.
URL: www.veritas.com/news/press/PressReleaseDetail.jhtml?NewsId=62911.

* * *

MicroStrategy is shipping MicroStrategy 7i Universal Edition for AIX 5L. Using the product, which provides 64-bit reporting, AIX 5L users will be able to analyse their corporate data and make business decisions.

The analytical engines provide different styles of business intelligence, including reporting, *ad hoc* queries, OLAP, ROLAP, and statistical analysis.

For further information contact:
MicroStrategy, 1861 International Drive, McLean, VA 22102, USA.
Tel: (703) 848 8600.
URL: www.microstrategy.com/Software/index.asp.

* * *

Peregrine Systems has introduced Peregrine Asset Tracking, which is designed to help businesses manage and consolidate views into their hardware, software, and network assets.

Peregrine Asset Tracking comprises Peregrine's AssetCenter Version 4.3 and a new discovery product bundle, Peregrine Enterprise Discovery, which combines functions of Peregrine Desktop Inventory and Peregrine Network Discovery.

Asset Tracking inventories IT assets, reveals configurations, and reconciles resources into a single portfolio. Asset Tracking includes application recognition for AIX, Windows, OS/2, Linux, Solaris, HP, and Java.

For further information contact:
Peregrine Systems, 3611 Valley Centre Drive, San Diego, CA 92130, USA.
Tel: (858) 481 5000.
URL: www.peregrine.com/asset-solutions.