222

MVS update

March 2005

In this issue

3 Introduction to z/OS Unix processing using REXX TSO functions

10 A background ANTRQST alternative to PPRC CQUERY commands

21 Using DFSORT to reformat records

25 Dataset performance reporter

47 Repairing a full VTOC index in an MVS DASD subsystem

50 Catching up with COBOL

58 Job scheduling isn't just for job schedulers

67 Workload management

75 MVS news

A xephon PUBLICATION

Published by

Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

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

Editor

Trevor Eddolls
E-mail: trevore@xephon.com

Publisher

Colin Smith
E-mail: info@xephon.com

Subscriptions and back-issues

A year's subscription to *MVS Update*, comprising twelve monthly issues, costs $505.00 in the USA and Canada; £340.00 in the UK; £346.00 in Europe; £352.00 in Australasia and Japan; and £350.00 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 2000 issue, are available separately to subscribers for £29.00 ($43.50) each including postage.

MVS Update on-line

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

Contributions

When Xephon is given copyright, articles published in *MVS 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.

Introduction to z/OS Unix processing using REXX TSO functions

This article details how to utilize REXX to perform Unix services. There is now a set of z/OS Unix extensions to the TSO/E Restructured Extended Executor (REXX) language. These enable REXX programs to access z/OS Unix callable services. These extensions are known as syscall commands and have names that correspond to the names of the callable services they invoke, eg chown, access, chmod, etc.

You can run an interpreted or compiled REXX program with syscall commands from TSO/E, from MVS batch, from the z/OS shells, or from a program.

You can run a REXX program with syscall commands only on a system with z/OS Unix System Services installed, and the correct level of RACF authorization in place.

REXX uses a number of Host Command Environments to implement these function calls. These environments, along with their descriptions, are shown below:

- SYSCALL – for a REXX program with syscall commands that will be run from TSO/E or MVS batch, you need to initialize the environment by beginning a REXX program with a syscalls('ON') call.
- SH – for a REXX program with syscall commands that will be run from a z/OS shell or from a program, SH is the initial host environment. The SYSCALL environment is automatically initialized as well, so you do not need to begin the REXX program with a syscalls('ON') call. Syscall commands within the REXX program (for example, chown) are interpreted as z/OS shell commands, not as syscall commands.
- TSO – a REXX program can run TSO/E commands, but you cannot use TSO commands to affect your REXX

environment, or have REXX statements or other host command environments affect your TSO process. Commands that are addressed to TSO will be run in a TMP running in a separate address space and process from your REXX program. The TSO process is started when the first TSO command is run, and persists until your REXX program terminates or you run the TSO LOGOFF command.

It is important to understand that restrictions exist on what can be done. If a REXX program is executed from a z/OS shell or from a program, SH and SYSCALL host command environments are available to it. If a REXX program is run from TSO/E or MVS batch, only the SYSCALL environment is available.

In TSO/E or MVS batch, use the syscalls('ON') function at the beginning of the REXX program. This will ensure that the SYSCALL command environment (ADDRESS syscall) is established and that the address space is a process. This is referred to as dubbing. It will also initialize the predefined variables in the current REXX variable pool. As well as this, it sets the signal process mask to block all signals that can be blocked and will clear the _ _argv. and _ _environment. stems.

There are four variations of the syscalls() function. They are:

- syscalls('ON') to establish the SYSCALL environment.
- syscalls('OFF') to end the SYSCALL environment.
- syscalls('SIGON') to establish the signal interface routine.
- syscalls('SIGOFF') to delete the signal interface routine.

You have to code ON, OFF, SIGON, and SIGOFF in upper case.

When you issue sysclass('ON'), one of the following return codes is set:

- 0 – successful completion.

- 4 – the signal process mask was not set.
- 7 – the process was dubbed, but the SYSCALL environment was not established.
- 8 – the process could not be dubbed.

Below is a sample program that shows how you can issue this function and test the return code:

```
Select
   When syscalls('ON') = Ø call retcodeØ
   When syscalls('ON') = 4 call retcode4
   When syscalls('ON') = 7 call retcode7
   When syscalls('ON') = 8 call retcode8
Otherwise
    Say 'Invalid return code found'
End
```

The SH environment is the default host command environment when a REXX program is run from a z/OS shell or from a program using exec(). It is available to a REXX program only in those two situations. In the SH environment, a syscall command runs as a z/OS shell command that has been issued this way:

```
/bin/sh -c shell_command
```

If you are running the REXX program from a z/OS shell or from a program, the SYSCALL environment is automatically initialized.

A REXX program can execute from the z/OS shell, or you can call it from any program just as you would call an executable program. The REXX program will execute as a separate process and not within the TSO/E address space. You cannot use TSO/E commands in the REXX program.

A REXX program that is invoked from a z/OS shell or from another program must be a text file or a compiled REXX program that resides in the hierarchical file system. It must have read and execute access permissions associated with it, and every line in the text file must be terminated by a newline character and must not exceed 2048 characters. Note that

sequence numbers are not supported, so ensure that sequence numbering is set to OFF when editing the program. If you are working in a z/OS shell environment and use only a filename to invoke the REXX program, the PATH environment variable is used to locate it.

The TSO command environment can also be used from a z/OS Unix REXX environment. It is initialized as expected by coding:

```
address tso
```

Commands addressed to TSO are run in a TSO TMP that is running in a separate address space and process from your REXX program. This provides you with the ability to run TSO commands. It does not provide you with the ability to use TSO commands to affect your REXX environment, or to have REXX statements or other host command environments affect your TSO process.

The TSO process is started when the first TSO/E command is run, and will remain in existence until your REXX program terminates or a TSO LOGOFF is issued. You can use the ps shell command to observe this process, shown as program bpxwrtso. Unexpected termination of the TSO process causes the next TSO command to fail with return code 16. You need to take care when processing TSO command input. Most TSO commands, including commands that prompt for missing arguments, use the TGET macro instruction for input. This results in a command error, and the command usually terminates. For commands that are able to read input, the source of the input is, first, any data that is currently on your stack, and then any data in your REXX program's standard input stream. Regardless of whether the command processes input, all data on the stack is queued to the TSO command. The stack is empty after any TSO command has been run. You need to be aware of this if you are using the stack to store information, because it could be lost.

The standard input stream may also be queued as input to the TSO command. For example, if you have a file redirected as

input and you run a TSO command before processing that file, some or all of the file may be queued to the TSO command. If input is the terminal, queued input may be queued to the TSO command. This characteristic can be used to interact with some TSO commands.

It is possible to turn off TSO command input by using the rexxopt() function with NOTSOIN coded. You can also use outtrap() to parse TSO command output into variables for later processing.

As you would expect, a REXX program can perform input and output. For a REXX program that is run from a z/OS shell or from a program, file descriptors 0, 1, and 2 (conventionally, standard input, standard output, and standard error files) are typically inherited. A read or write error on file descriptors 0, 1, or 2 will end up with a halt interruption if the read or write was from a PARSE EXTERNAL instruction, a SAY instruction, or EXECIO.

If the REXX program issues a PARSE EXTERNAL instruction, it reads standard input for a single text record. The newline character is removed from the record before it is returned to the REXX program. Standard input is assumed to be a text file, such as your terminal input. If the REXX program issues a SAY instruction, the text is directed to standard output, and a newline character is appended to the end of the text. Messages issued by REXX, including error and trace messages, are similarly directed to standard output.

The SYSCALL host command environment gives you more direct control over input and output. REXX can use the readfile and writefile read and write calls.

The REXX signal services consist of the following syscall commands:

- alarm
- kill
- pause

- sigaction
- sigpending
- sigprocmask
- sigsuspend
- sleep.

IBM has provided a way to customize the REXX environment. When a REXX program is run from the z/OS shells or called from a program using exec(), the z/OS Unix REXX environment that is established is created from the module BPXWRXEV. The source for this module is contained as a member named BPXWRX01 in SYS1.SAMPLIB.

This environment is inherited from the default MVS REXX environment.

However, the default handling of error messages from the REXX processor is overridden so that the messages are written to STDOUT. This is the same place to which output from the SAY instruction and trace information is sent. Further customization can be achieved by altering this sample member.

The syscalls('ON') function ensures that the SYSCALL host command environment is available in your REXX environment. If a call detects that SYSCALL is not available in your environment, it dynamically adds it. However, performance characteristics for dynamically-added host commands are not as good as for host commands that are included in the initial environment. This is because every time a command is directed to the SYSCALL host command environment, the TSO/E REXX support loads the module for the SYSCALL host command. This can be avoided by including the syscall host command in three default TSO/E environments. These environments and the members in SYS1.SAMPLIB that define them are shown below:

- TSO – IRXTSPRM, IRXREXX2
- MVS – IRXPARMS, IRXREXX1

- ISPF – IRXISPRM, IRXREXX3.

By customizing IRXxxPRM, you will provide performance improvement for REXX programs that use syscall commands from TSO/E or MVS batch.

Make the following changes to the SYS1.SAMPLIB members to add the SYSCALL host command to that default environment:

- Find the label SUBCOMTB_TOTAL and add 1 to its value. For example, change SUBCOMTB_TOTAL DC F'14' to SUBCOMTB_TOTAL DC F'15'.
- Find the label SUBCOMTB_USED and add 1 to its value. For example, change SUBCOMTB_USED DC F'14' to SUBCOMTB_USED DC F'15'.
- Find the end of the subcommand table, just before the label PACKTB or PACKTB_HEADER, and add the following lines:

```
SUBCOMTB_NAME_REXXIX    DC  CL8'SYSCALL '
SUBCOMTB_ROUTINE_REXXIX DC  CL8'BPXWREXX'
SUBCOMTB_TOKEN_REXXIX   DC  CL16' '
```

- Assemble and link-edit the module and replace the default TSO/E module. These are normally in SYS1.LPALIB.

Full details on the functions described here along with references to all syscall command operands can be found in the IBM manual *Using REXX and z/OS UNIX System Services* – SA22-7806-03.

John Bradley
Systems Programmer
Meerkat Computer Services (UK)

A background ANTRQST alternative to PPRC CQUERY commands

BACKGROUND

Like many other sites, we have implemented a disaster recovery strategy that consists in splitting processors, DASD subsystems, tape subsystems, Sysplex Timers, etc across physical locations. In our case we have our hardware resources split between a local and a remote site and the infrastructure has been designed to allow access to all peripherals from either processor.

We have implemented IBM's Peer-to-Peer Remote Copy (PPRC) hardware solution to facilitate rapid and accurate disaster recovery (as well as the additional benefits of allowing workload and DASD migration). PPRC is based on the 3990 storage controller and comprises two similar DASD subsystems connected to each other by one or more ESCON links. Updates made on the primary DASD volumes (in our case the local subsystem) are synchronously shadowed (ie mirrored) to the secondary DASD volumes (in our case the remote subsystem). With PPRC no DASD data is lost between the last update at the primary subsystem and recovery at the DR site.

A detailed PPRC implementation plan is beyond the scope of this article, but I would suggest that a good starting place is *z/OS V1R3.0 DFSMS Advanced Copy Services*, which is available on the IBM Web site. Briefly, however, assuming that the necessary hardware and software prerequisites are in place, you issue the CESTPATH command to establish paths between the primary and secondary storage controls. You then issue the CESTPAIR command to establish the mirrored pairs. It's as simple as that.

THE CQUERY COMMAND

It is possible to query the status of one volume of a PPRC pair by issuing the CQUERY command. This command has the format:

```
TSO CQUERY DEVN(X'devn')
```

where *devn* is the four-digit hexadecimal address of the devices to which the I/O operation is directed.

The output of this command is sent to the user's TSO session and is also put in the SYSLOG, and can be viewed via SDSF or a compatible product. On our system, issuing:

```
TSO CQUERY DEVN(X'1205')
```

produces the following output:

```
ANTP0090I CQUERY FORMATTED LVL 3
VOLUME REPORT
************** PPRC REMOTE COPY CQUERY - VOLUME ********************
*                                          (PRIMARY)   (SECONDARY) *
*                                         SSID CCA LSS SSID CCA LSS*
*DEVICE   LEVEL      STATE     PATH STATUS  SERIAL#     SERIAL#    *
*------- --------  ----------  ------------ --------    ---------  *
* 1205  PRIMARY..  DUPLEX....   ACTIVE..   1004 05 04   2004 05 04 *
*       CRIT(NO).......       CGRPLB(NO). 000000012345 000000067890*
* PATHS SAID DEST STATUS: DESCRIPTION                              *
* ----- --------- ------  -----------------                       *
*   4   0000 0000   01    PATH ESTABLISHED...                     *
*       0020 0000   01    PATH ESTABLISHED...                     *
*       0080 0000   01    PATH ESTABLISHED...                     *
*       00A0 0000   01    PATH ESTABLISHED...                     *
********************************************************************
ANTP0001I CQUERY COMMAND COMPLETED FOR DEVICE 1205. COMPLETION CODE: 00
```

The output above shows the status for device number 1205. You will see that it is Channel Connection Address (CCA) 05 on Logical Subsystem (LSS) 04 of Subsystem Identification (SSID) 1004, which has a serial number of 12345. This device is the primary volume in a duplexed (ie mirrored) pair and its secondary volume has CCA 05, LSS 04, SSID 2004, and serial number 67890. This means that all updates to this local volume are automatically mirrored to the paired remote volume providing an identical copy for DR purposes. For simplicity we

keep device number 1205 paired with 2205, 1234 paired with 2234 etc because it makes it much easier to figure out what is going on!

Issuing TSO CQUERY DEVN(X'2205') shows that device number 2205 is a secondary volume in a duplexed pair. Note that the path information is only shown for a primary volume:

```
ANTP0090I CQUERY FORMATTED LVL 3
VOLUME REPORT
************** PPRC REMOTE COPY CQUERY - VOLUME ********************
*                                          (PRIMARY)   (SECONDARY) *
*                                         SSID CCA LSS SSID CCA LSS*
*DEVICE   LEVEL      STATE     PATH STATUS  SERIAL#     SERIAL#    *
*------- --------  ----------  ------------ --------    ---------  *
* 2205  SECONDARY  DUPLEX....   ACTIVE..   1004 05 04   2004 05 04 *
*       ..............       ..........  000000012345 000000067890*
* PATHS SAID DEST STATUS: DESCRIPTION                               *
* ----- --------- ------  -----------------                        *
*   0   ---- ----   00    NO PATH............                      *
*       ---- ----   00    NO PATH............                      *
*       ---- ----   00    NO PATH............                      *
*       ---- ----   00    NO PATH............                      *
*********************************************************************
ANTP0001I CQUERY COMMAND COMPLETED FOR DEVICE 2205. COMPLETION CODE: 00
```

CQUERY DRAWBACKS

That's all very well, but there is a downside to using CQUERY. One limitation is that the command can be issued for only one device at a time.

If you want to query the status of more that one volume, you need to repeat the command serially for each volume or, preferably, run TSO under batch and input all the commands one after another as the following example shows:

```
//... jobcard goes here
//*
//CQUERY  EXEC PGM=IKJEFT01,DYNAMNBR=30
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
  CQUERY DEVN(X'1200')
  CQUERY DEVN(X'1201')
  CQUERY DEVN(X'1202')
  ...... ............
```

```
  CQUERY DEVN(X'120F')
/*
//
```

Even though this allows us to query the status of multiple devices it is still a laborious, time-consuming, and error-prone job to set up the JCL when there are many devices to be queried. Even though we have IBM's excellent ISPF/PDF editing facilities to help out, I can think of better things to do than sit and build a job for a DASD subsystem containing 4096 devices! This is one area in which CQUERY is somewhat inflexible.

Another problem is that all the command output is routed to the syslog and, if the ANTP0090I messages are not suppressed using MPF (or one of the many automation products on the market), the MVS operator console can be flooded with messages. As we know, this could lead to a WTO buffer shortage and, if things aren't resolved quickly, an unscheduled IPL. Try explaining that one to management during prime shift!

A final problem is that the output is verbose and takes up a lot of space for the information it shows. It is not always easy to look at the output and obtain the important information that you are looking for.

THE PPRCSTAT PROGRAM

I realized that it would be much simpler if I could write a program that would provide the required information for a single device, or a range of devices, and write the output to a sequential dataset or the JES spool. I dug about in the *z/OS V1R3.0 DFSMSdfp Advanced Services* manual and came across the ANTRQST macro. This macro provides an application program call to the z/OS System Data Mover's (SDM) Application Programming Interface (API) and allows you to call XRC, PPRC, FlashCopy, and Snapshot Copy functions. This seemed like what I wanted!

I wrote the PPRCSTAT Assembler program, which is run as a

batch job, to issue the ANTRQST PQUERY command for a single device or a range of devices. The PPRCSTAT program should be assembled and link-edited, with AC(1), to an APF-authorized library in the normal manner. The program must run authorized because it uses the UCBLOOK macro.

CODE

```
         TITLE 'PPRCSTAT: ISSUE ANTRQST PQUERY COMMAND'
PPRCSTAT CSECT ,
PPRCSTAT AMODE 31
PPRCSTAT RMODE 24
         PRINT NOGEN
         SAVE  (14,12),,PPRCSTAT_&SYSDATE._&SYSTIME SAVE REGISTERS
         USING PPRCSTAT,R12            ESTABLISH ADDRESSABILITY
         LR    R12,R15                 LOAD BASE REG WITH ENTRY POINT
         LR    R15,R13                 LOAD CALLER'S SAVE AREA ADDRESS
         ST    R13,SAVE+4              SAVE CALLER'S SAVE AREA ADDRESS
         LA    R13,SAVE                LOAD OUR SAVE AREA ADDRESS
         ST    R13,8(R15)              SAVE OUR SAVE AREA ADDRESS
         L     R1,Ø(R1)                PICK UP PARM
         LH    R2,Ø(R1)                GET PARM LENGTH
         LA    R3,2(R1)                GET ADDRESS OF PARM DETAILS
         CH    R2,=H'9'                CHECK LENGTH OF PARM
         BH    PARMERR                 LENGTH GT 9, SHOW ERROR
         BE    PARM9                   LENGTH EQ 9, CONTINUE
         CH    R2,=H'4'                LENGTH = 4 ?
         BNE   PARMERR                 N, SHOW ERROR
         MVI   4(R3),C','              MOVE IN COMMA
         MVC   5(4,R3),Ø(R3)           COPY START DEV NO TO END DEV NO
         LA    R2,5(R2)                INCREASE PARM LENGTH BY 5
PARM9    DS    ØH
         CLI   4(R3),C','              COMMA BETWEEN DEV NOS ?
         BNE   NOCOMMA                 N, SHOW ERROR
         TR    5(4,R3),TRTAB           TRANSLATE A...F TO X'FA...FF'
         PACK  OUTPUT(3),5(4,R3)       PACK END DEVICE NUMBER
         L     R4,OUTPUT               GET PACKED DATA
         SRL   R4,12                   DROP LAST 3 NIBBLES
         LR    R7,R4                   SAVE END HEX DEVICE NUMBERS
         TR    Ø(4,R3),TRTAB           TRANSLATE A...F TO X'FA...FF'
         PACK  OUTPUT(3),Ø(4,R3)       PACK START DEVICE NUMBER
         L     R4,OUTPUT               GET PACKED DATA
         SRL   R4,12                   DROP LAST 3 NIBBLES
         CR    R4,R7                   START DEVICE NUMBER BIGGER ?
         BH    DEVN1BIG                Y, SHOW ERROR
         LA    R6,1(RØ)                SET INCREMENT TO 1
         OPEN  (OUTFILE,(OUTPUT))      OPEN OUTFILE
```

```
         LTR   R15,R15                SUCCESSFUL OPEN ?
         BNZ   BADOPEN                N, SHOW ERROR
ANTLOOP  DS    ØH
         STH   R4,XDEVN               SAVE START HEX DEVICE NUMBER
         LA    RØ,L'XQRYINFO          GET SIZE OF XQRYINFO
         STH   RØ,XQRYSIZE            AND SAVE IT
         ANTRQST ILK=PPRC,            PPRC FUNCTION                    *
               REQUEST=PQUERY,        PQUERY                           *
               DEVN=XDEVN,            DEVICE NUMBER                    *
               QRYSIZE=XQRYSIZE,      QRYINFO SIZE                     *
               QRYINFO=XQRYINFO,      OUTPUT INFO AREA                 *
               RETINFO=XRETINFO,      RET CODE INFO AREA               *
               RETCODE=RTNCD,         RETURN CODE                      *
               RSNCODE=RSNCD,         REASON CODE                      *
               MF=(E,P_LIST)          EXECUTE FORM
         LTR   R15,R15                ANTRQST PARM ERROR ?
         BNZ   ANTRQ_PARMERR          Y, SHOW ERROR
         L     R15,RTC                LOAD RETINFO RETURN CODE
         L     RØ,RSN                 LOAD RETINFO REASON CODE
         C     R15,=A(RQST_PQUERY_QRYSIZE_BIG_ENOUGH) BIG ENOUGH ?
         BE    BIGENUFF               Y, CONTINUE
         C     R15,=A(RQST_PC_NUMBER_ZERO) ANTASØØØ STARTED ?
         BE    NOTSTART               N, SHOW ERROR
         C     R15,=A(RQST_PQUERY_ERROR) SDM ERROR 724Ø ?
         BNE   ANTRQ_QRYERR           N, SHOW ERROR
         CLC   =CL4'213I',ANTMSG      ESS PAV DEVICE ?
         BE    LOOPCNTL               Y, IGNORE IT AND PROCESS NEXT
         B     ANTRQ_ERROR            N, SHOW ERROR
BIGENUFF DS    ØH
         MVI   BUFFER,C' '            BLANK OUT FIRST CHARACTER
         MVC   BUFFER+1(79),BUFFER    PROPAGATE THROUGH BUFFER
         LA    R11,XQRYINFO           POINT TO ANTRQST INFO
         CLI   XQRYINFO+39,C','       IS IT AN SVA ? IE NO LSSID
         BE    MOVESVA                Y, MOVE IN SVA DETAILS
         USING EQUERYD,R11            ESTABLISH ADDRESSABILITY
         MVC   BDEVN,EDEVN            MOVE IN DEVICE NUMBER
         MVC   BLEVEL,ELEVEL          MOVE IN LEVEL
         MVC   BSTATE,ESTATE          MOVE IN PAIR STATE
         MVC   BPTHSTAT,EPTHSTAT      MOVE IN PATH STATUS
         MVC   BPSSID,EPSSID          MOVE IN PRIMARY SSID
         MVC   BPCCA,EPCCA            MOVE IN PRIMARY CCA
         MVC   BPLSSID,EPLSSID        MOVE IN PRIMARY LSSID
         MVC   BSSSID,ESSSID          MOVE IN SECONDARY SSID
         MVC   BSCCA,ESCCA            MOVE IN SECONDARY CCA
         MVC   BSLSSID,ESLSSID        MOVE IN SECONDARY LSSID
         B     GETVOL                 GO AND GET VOLSER FROM UCB
MOVESVA  DS    ØH
         USING SQUERYD,R11            ESTABLISH ADDRESSABILITY
         MVC   BDEVN,SDEVN            MOVE IN DEVICE NUMBER
         MVC   BLEVEL,SLEVEL          MOVE IN LEVEL
```

```
         MVC   BSTATE,SSTATE           MOVE IN PAIR STATE
         MVC   BPTHSTAT,SPTHSTAT       MOVE IN PATH STATUS
         MVC   BPSSID,SPSSID           MOVE IN PRIMARY SSID
         MVC   BPCCA,SPCCA             MOVE IN PRIMARY CCA
         MVC   BSSSID,SSSSID           MOVE IN SECONDARY SSID
         MVC   BSCCA,SSCCA             MOVE IN SECONDARY CCA
GETVOL   DS    ØH
         MODESET MODE=SUP              GET INTO SUPERVISOR MODE
         UCBLOOK DEVN=XDEVN,           GET UCB FOR BINARY DEVICE NO.   *
               UCBPTR=UCBPTR,          --> UCB COMMON SECTION          *
               DYNAMIC=YES,            INCLUDE DYNAMIC UCBS            *
               LOC=ANY,                INCLUDE UCBS ABOVE 16MB         *
               RANGE=ALL,              INCLUDE 4DIGIT UCBS             *
               NOPIN,                  DON'T PIN UCB                   *
               RETCODE=RTNCD,          RETURN CODE                     *
               RSNCODE=RSNCD           REASON CODE
         MODESET MODE=PROB             GET INTO PROBLEM MODE
         MVC   BVOLSER,NOVOLSER        PRIME WITH UNKNOWN VOLSER
         LTR   R15,R15                 RCØ FROM UCBLOOK ?
         BNZ   PUT                     N, GO AND SHOW INFO
         L     R5,UCBPTR               GET POINTER TO UCB COMMON AREA
         USING UCBCMSEG,R5             ESTABLISH ADDRESSABILITY
         MVC   BVOLSER,OFFLINE         PRIME WITH OFFLINE VOLSER
         CLC   UCBVOLI,HEXZERO6        OFFLINE ?
         BE    PUT                     Y, GO AND SHOW INFO
         MVC   BVOLSER,UCBVOLI         MOVE IN VOLSER
PUT      DS    ØH
         PUT   OUTFILE,BUFFER          WRITE OUT PARM DETAILS
LOOPCNTL DS    ØH
         BXLE  R4,R6,ANTLOOP           LOOP FOR ALL DEV NOS IN RANGE
         CLOSE OUTFILE                 CLOSE OUTFILE
         LTR   R15,R15                 SUCCESSFUL CLOSE ?
         BNZ   BADCLOSE                N, SHOW ERROR
         B     RETURN                  AND EXIT
*
RETURN   DS    ØH
         L     R13,4(,R13)             RESTORE CALLER'S SAVE AREA ADDR
         RETURN (14,12),RC=Ø           AND RETURN
*
ANTRQ_PARMERR  EQU  *
         MVC   WTOS(MSG1LN),MSG1           MOVE IN WTO MSG
         CVD   R15,WA                      CONVERT RET CODE TO DECIMAL
         MVC   WTOS+£I1(L'EDMASKT),EDMASKT MOVE IN EDIT PATTERN
         ED    WTOS+£I1(L'EDMASKT),WA+5    EDIT RETURN CODE
         CVD   RØ,WA                       CONVERT RSN CODE TO DECIMAL
         MVC   WTOS+£I2(L'EDMASKT),EDMASKT MOVE IN EDIT PATTERN
         ED    WTOS+£I2(L'EDMASKT),WA+5    EDIT REASON CODE
         WTO   MF=(E,WTOS)                 ISSUE WTO
         B     RETURN                      AND EXIT
*
```

```
ANTRQ_QRYERR   EQU  *
         MVC   WTOS(MSG2LN),MSG2           MOVE IN WTO MSG
         CVD   R15,WA                      CONVERT RET CODE TO DECIMAL
         MVC   WTOS+£I1(L'EDMASKT),EDMASKT MOVE IN EDIT PATTERN
         ED    WTOS+£I1(L'EDMASKT),WA+5    EDIT RETURN CODE
         CVD   RØ,WA                       CONVERT RSN CODE TO DECIMAL
         MVC   WTOS+£I2(L'EDMASKT),EDMASKT MOVE IN EDIT PATTERN
         ED    WTOS+£I2(L'EDMASKT),WA+5    EDIT REASON CODE
         WTO   MF=(E,WTOS)                 ISSUE WTO
         B     RETURN                      AND EXIT
*
ANTRQ_ERROR    EQU  *
         ICM   R9,B'1111',ANTMSGL      MOVE IN MSG LENGTH
         SRL   R9,24                   MOVE TO LAST BYTE OF R9
         LTR   R9,R9                   ZERO MSG LENGTH ?
         BZ    RETURN                  Y, EXIT
         BCTR  R9,Ø                    REDUCE LENGTH FOR EXECUTE
         MVC   WTOS(MSGALN),WTOANT     MOVE IN MSG
         LA    R8,WTOS+£I3             POINT PAST ANTPØ IN MSG
         EX    R9,MVCANT               MOVE IN ANTMSG
         WTO   MF=(E,WTOS)             ISSUE WTO
         B     RETURN                  AND EXIT
*
PARMERR  DS    ØH
         WTO   'PPRCSTAT: PARM MUST BE 4 OR 9 BYTES IN LENGTH',        *
               ROUTCDE=11              ISSUE WTO
         B     RETURN                  AND EXIT
*
NOCOMMA  DS    ØH
         WTO   'PPRCSTAT: PARM MUST HAVE COMMA BETWEEN DEV NOS',       *
               ROUTCDE=11              ISSUE WTO
         B     RETURN                  AND EXIT
*
DEVN1BIG DS    ØH
         WTO   'PPRCSTAT: START DEV NO GREATER THAN END DEV NO',       *
               ROUTCDE=11              ISSUE WTO
         B     RETURN                  AND EXIT
*
BADOPEN  DS    ØH
         WTO   'PPRCSTAT: UNSUCCESSFUL OPEN OF OUTFILE',               *
               ROUTCDE=11              ISSUE WTO
         B     RETURN                  AND EXIT
*
BADCLOSE DS    ØH
         WTO   'PPRCSTAT: UNSUCCESSFUL CLOSE OF OUTFILE',              *
               ROUTCDE=11              ISSUE WTO
         B     RETURN                  AND EXIT
*
NOTSTART DS    ØH
         WTO   'PPRCSTAT: ANTASØØØ MUST BE STARTED',                   *
```

```
               ROUTCDE=11              ISSUE WTO
          B    RETURN                  AND EXIT
*
MSG1      WTO  'PPRCSTAT:  PARM ERROR: RC=Ø12345 RSN=Ø12345',          *
               ROUTCDE=11,MF=L
MSG1LN    EQU  *-MSG1
*
MSG2      WTO  'PPRCSTAT: QUERY ERROR: RC=Ø12345 RSN=Ø12345',          *
               ROUTCDE=11,MF=L
MSG2LN    EQU  *-MSG2
*
WTOANT    WTO  'ANTPØ                                                  *
                                                                    ',*
               ROUTCDE=11,MF=L
MSGALN    EQU  *-WTOANT
*
OUTFILE   DCB  DSORG=PS,DDNAME=OUTFILE,MACRF=PM,LRECL=8Ø,              *
               BLKSIZE=312Ø,RECFM=FB
*
          DS   ØD
TRTAB     DC   XL256'ØØ'
          ORG  TRTAB+X'81'            ORG TO LOWERCASE 'A'
          DC   X'FAFBFCFDFEFF'        TRANSLATE LOWERCASE 'ABCDEF'
          ORG  TRTAB+C'A'             ORG TO UPPERCASE 'A'
          DC   X'FAFBFCFDFEFF'        TRANSLATE UPPERCASE 'ABCDEF'
          ORG  TRTAB+C'Ø'
          DC   C'Ø123456789'
          ORG
          DS   ØD
*
          LTORG
*
          REGEQU ,                    REGISTER EQUATES
*
£I1       EQU  3Ø                     OFFSET FOR RTN CODE IN WTOS
£I2       EQU  41                     OFFSET FOR RSN CODE IN WTOS
£I3       EQU  9                      OFFSET FOR ANTPØ MSG IN WTOS
MVCANT    MVC  Ø(*-*,R8),ANTMSG       MOVE IN ANTMSG
EDMASKT   DC   X'4Ø2Ø2Ø2Ø212Ø'        EDIT MASK FOR RTN/RSN CODE
SAVE      DS   18F                    SAVE AREA
OUTPUT    DS   F                      WORK AREA FOR PACK
WA        DS   D                      WORK AREA FOR CVD
WTOS      DS   XL128                  WORK AREA FOR WTO
UCBPTR    DS   CL4                    POINTER TO UCB COMMON SECTION
NOVOLSER  DC   CL6'??????'            USED ON UCBLOOK FAILURE
OFFLINE   DC   CL6'*OFFL*'            OFFLINE VOLUME INDICATOR
HEXZERO6  DC   XL6'ØØØØØØØØØØØØ'      OFFLINE VOLUME IN UCBVOLI
RTNCD     DS   F                      RETURN CODE
RSNCD     DS   F                      REASON CODE
DEVN1     DS   H                      START HEX DEVICE NUMBER
```

```
DEVN2    DS    H                        END HEX DEVICE NUMBER
XDEVN    DS    H                        ANTRQST INPUT HEX DEVICE NUMBER
XQRYSIZE DS    H                        ANTRQST SIZE OF OUTPUT INFO AREA
XQRYINFO DS    CL512                    ANTRQST OUTPUT INFO AREA
         DS    ØF
XRETINFO DS    CL1ØØ                    ANTRQST RETURN CODE INFO
         ORG   XRETINFO
RTC      DS    F                        ANTRQST RETURN CODE
RSN      DS    F                        ANTRQST REASON CODE
ANTMSGL  DS    XL1                      ANTRQST MSG LENGTH
ANTMSG   DS    ØX                       ANTRQST MSG
         ORG   XRETINFO+L'XRETINFO
         ANTRQSTL NAME=P_LIST,BASE=ØF
*
BUFFER   DS    ØCL8Ø                    PRINT LINE
BDEVN    DS    CL4                      DEVICE NUMBER
         DS    CL2
BLEVEL   DS    CL9                      LEVEL (PRIMARY/SECONDARY)
         DS    CL2
BSTATE   DS    CL1Ø                     PAIR STATE (SIMPLEX/DUPLEX/
*                                                  COPYING/SUSPENDED)
         DS    CL3
BPTHSTAT DS    CL8                      PATH STATUS (ACTIVE/INACTIVE)
         DS    CL3
BPSSID   DS    CL4                      PRIMARY SSID
         DS    CL1
BPCCA    DS    CL2                      PRIMARY CCA
         DS    CL1
BPLSSID  DS    CL2                      PRIMARY LSSID
         DS    CL3
BSSSID   DS    CL4                      SECONDARY SSID
         DS    CL1
BSCCA    DS    CL2                      SECONDARY CCA
         DS    CL1
BSLSSID  DS    CL2                      SECONDARY LSSID
         DS    CL3
BVOLSER  DS    CL6                      VOLSER
         DS    CL7
*
*  STK SVA ANTRQST PQUERY DATA AREA (SEE ANTPØ91I FOR LAYOUT)
*
SQUERYD  DSECT
SDEVN    DS    CL4                      DEVICE NUMBER
         DS    CL1
SLEVEL   DS    CL9                      LEVEL (PRIMARY/SECONDARY)
         DS    CL1
SSTATE   DS    CL1Ø                     PAIR STATE (SIMPLEX/DUPLEX/
*                                                  COPYING/SUSPENDED)
         DS    CL1
SPTHSTAT DS    CL8                      PATH STATUS (ACTIVE/INACTIVE)
```

```
         DS    CL1
SPSSID   DS    CL4                     PRIMARY SSID
         DS    CL1
SPCCA    DS    CL2                     PRIMARY CCA
         DS    CL1
SPSERIAL DS    CL12                    PRIMARY SERIAL #
         DS    CL1
SSSSID   DS    CL4                     SECONDARY SSID
         DS    CL1
SSCCA    DS    CL2                     SECONDARY CCA
         DS    CL1
SSSERIAL DS    CL12                    SECONDARY SERIAL #
SQUERYDL EQU   *-SQUERYD
*
*  IBM ESS ANTRQST PQUERY DATA AREA (SEE ANTPØ91I FOR LAYOUT)
*
EQUERYD  DSECT
EDEVN    DS    CL4                     DEVICE NUMBER
         DS    CL1
ELEVEL   DS    CL9                     LEVEL (PRIMARY/SECONDARY)
         DS    CL1
ESTATE   DS    CL1Ø                    PAIR STATE (SIMPLEX/DUPLEX/
*                                                  COPYING/SUSPENDED)
         DS    CL1
EPTHSTAT DS    CL8                     PATH STATUS (ACTIVE/INACTIVE)
         DS    CL1
EPSSID   DS    CL4                     PRIMARY SSID
EPLSSID  DS    CL2                     PRIMARY LSS ID
         DS    CL1
EPCCA    DS    CL2                     PRIMARY CCA
         DS    CL1
EPSERIAL DS    CL12                    PRIMARY SERIAL #
         DS    CL1
ESSSID   DS    CL4                     SECONDARY SSID
ESLSSID  DS    CL2                     SECONDARY LSS ID
         DS    CL1
ESCCA    DS    CL2                     SECONDARY CCA
         DS    CL1
ESSERIAL DS    CL12                    SECONDARY SERIAL #
EQUERYDL EQU   *-EQUERYD
*
         IEFUCBOB ,                    UCB MAPPING MACRO
*
         END   PPRCSTAT
```

PPRCSTAT USAGE

The program has been tested with IBM's Enterprise Storage Server (ESS) and StorageTek's Shared Virtual Array (SVA)

DASD subsystems, but should, if required, be tailorable to other manufacturers' DASD. The program can be executed using the following JCL:

```
//... jobcard goes here
//*
//PPRC     EXEC PGM=PPRCSTAT,PARM='mmmm,nnnn'
//STEPLIB   DD  DSN=authlib,DISP=SHR  <=== required if not linklist'ed
//*UTFILE   DD  DSN=outfile,DISP=(NEW,CATLG),SPACE=(TRK,(1,1))
//OUTFILE   DD  SYSOUT=*
//
```

The parameter can be coded as 'mmmm' for one device, or 'mmmm,nnnn' for a range of devices where *mmmm* and *nnnn* are the 4-digit hexadecimal addresses of the devices to be queried. Running PPRCSTAT with PARM='1205,1208' gives the following output in OUTFILE:

```
1205  PRIMARY   DUPLEX     ACTIVE    1004 05 04   2004 05 04   AB1205
1206  PRIMARY   SUSPEND(3) ACTIVE    1004 06 04   2004 06 04   AB1206
1207            SIMPLEX    INACTIVE  1004 07 04                AB1207
1208            SIMPLEX    INACTIVE  1004 08 04                *OFFL*
```

From this you can see that device number 1205 is the primary volume in a duplexed pair, 1206 is the primary volume in a suspended duplex pair, 1207 is not PPRCed and on-line, and 1208 is not PPRCed and off-line.

You will, I hope, agree that PPRCSTAT is more flexible than the existing CQUERY solution and that it provides concise, accurate output without flooding the operator console with spurious messages.

Iain McArthur
Systems Programmer (UK)

Using DFSORT to reformat records

The operands INREC, OUTREC, and OUTFIL OUTREC can be used to reformat input records into different states for

output. You can define which input fields are included in the output record, in what order the input fields should appear, how they are aligned, and how they are edited or changed. Character and hexadecimal separators can be inserted before, between, and after the input fields.

INREC, OUTREC, and OUTFIL OUTREC can be used to generate constants for the current date and time in a number of different character, zoned decimal, and packed decimal formats. This allows you to add timestamps to records.

The following is an example of adding timestamps:

```
INREC FIELDS=(2X,DATE1(.),X,TIME1(:),X,1,50)
```

The output records for the SORTOUT dataset will contain the following:

1 Blanks in positions 1–2.

2 In output positions 3–12, the current date in the form ‘yyyy.mm.dd’.

3 A blank in position 13.

4 In output positions 14–21, the current time in the form C'hh:mm:ss'.

5 A blank in position 22.

6 In output positions 23–72, the characters from input positions 1–50.

INREC, OUTREC, and OUTFIL OUTREC let you display any field in a record in hexadecimal. For example:

```
OUTFIL OUTREC=(1,20,HEX,C'**',23,16,HEX)
```

The output records for the SORTOUT dataset will contain the following:

1 In output positions 1–20, the hexadecimal representation of input positions 1–20.

2 Two plus signs in output positions 21–22.

3 In output positions 23–38, the hexadecimal representation of input positions 21–36.

Translation of characters from lower-case EBCDIC letters to upper-case EBCDIC letters can also be performed. This is done using the TRAN= operand. In the following example, using INREC characters in positions 1 to 20 will have any lower-case characters changed to upper case. For example if the record contained ‘John Bradley’ it would be output as ‘JOHN BRADLEY’:

```
INREC FIELDS=(1,20,TRAN=LTOU)
```

You can also do the reverse and translate from upper-case EBCDIC letters to lower-case EBCDIC letters. In the following OUTREC this is shown:

```
OUTREC FIELDS=(1,4,5,TRAN=UTOL)
```

You can also perform specific translations using the ALTSEQ= operand. This defines an alternate sequence table. For example if you wanted to change X'00' to a space (X'40') you would code:

```
ALTSEQ CODE=(0040)
OUTFIL OUTREC=(1,20,21,40,TRAN=ALTSEQ)
```

In the above, output positions 1 to 20 will contain the characters from the input record positions 1 to 20. In output positions 21 to 60 the input record positions from 21 to 60 will be transferred, but if hexadecimal zeros are found they will be replaced with a space.

INREC, OUTREC, and OUTFIL OUTREC provide complex editing capabilities for controlling how numeric fields, including SMF date and time fields, and two-digit year date fields, are presented with respect to length, leading or suppressed zeros, thousands separators, decimal points, leading and trailing positive and negative signs, and so on. You can edit BI, FI, PD, PD0, ZD, CSF/FS, Y2x, DT1, DT2, DT3, TM1, TM2, TM3, and TM4 format fields. Twenty-seven pre-defined editing masks are available for commonly-used numeric patterns,

encompassing many of the numeric notations used throughout the world. In addition, a virtually unlimited number of numeric editing patterns are available with user-defined editing masks. Editing capabilities are particularly useful for OUTFIL reports.

For example look at the following:

```
OUTFIL FNAMES=(SYSOUT1,BACKUP1),
            OUTREC=(5:21,7,ZD,M18,25:46,3,ZD,M12,
            51:8,3,PD,EDIT=(IT.TTT))
```

SYSOUT1 and BACKUP1 output records will contain:

1 Blanks in output positions 1–4.

2 In output positions 5–14, the ZD value in input positions 21–27 is converted to a pattern of SII,IIT.TT, where *S* represents either a blank for plus or - sign for minus, *I* represents a leading insignificant digit, and *T* represents a significant digit.

3 Blanks in output positions 15–24.

4 In output positions 25–29, the ZD value in input positions 46–48 is converted to a pattern of ST.TT, where *S* represents either a blank for plus or - sign for minus and *T* represents a significant digit.

5 Blanks in output positions 30–50.

6 In output positions 51–56, the PD value in input positions 8–10 is converted to a pattern of IT.TTT, where *I* represents a leading insignificant digit and *T* represents a significant digit.

Again INREC, OUTREC, and OUTFIL OUTREC let you generate BI, PD, ZD, or CSF/FS sequence numbers in the output records. Starting values and increment values can be specified or allowed to default. The sequence numbers are assigned in the order in which the records are received for INREC, OUTREC, or OUTFIL OUTREC processing. Depending on the operand being used, processing is slightly different. If INREC is used, sequence numbers are assigned

before sorting, copying, or merging; while with OUTREC, the sequence numbers are assigned after sorting, merging, or copying. With OUTFIL OUTREC, the sequence numbers are assigned to each OUTFIL record, so records for different OUTFIL datasets can have the same sequence numbers.

DFSORT's formatting capabilities are endless and remove the need for complex or simple programs to perform record operations.

John Bradley
Systems Programmer
Meerkat Computer Services (UK)

Dataset performance reporter

INTRODUCTION

In a contemporary rapidly-changing IT environment, where a new application should be designed, developed, tested, and implemented in the shortest time possible, or modified soon after, also quickly, it has become increasingly difficult to maintain a stable and consistent application environment. Although new equipment can be installed at a reduced per MIPS price, applications are consuming the hardware at a steadily increasing pace. Thus, the MVS environment I/O activity tuning opportunities can produce results that may seem like miracles. The productive capacity of a complex can be greatly increased because I/O activity tuning will also improve processor, main storage, and virtual storage utilization.

As is well known, the life of data begins and ends on a disk volume (except for some rare data-in-memory applications). When data has to be placed on or retrieved from a disk volume, it takes many orders of magnitude longer than any

other component of the total service time. When we break down an application's response time into components, we often find that the majority of the time (between 60% and 70%) is spent performing I/O. If an application is I/O-bound, the I/O may take up to 90% of the elapsed time, and even if it is CPU-bound the I/O may still be between 30% and 40% of the total elapsed time. Typically, I/O time is 60–90% of the total time.

I/O ANALYSIS

There are a number of ways to reduce the elapsed time by improving I/O performance, but the very first step in all cases is to perform an analysis of I/O. The I/O analysis process in itself is often a combination of one or more approaches, depending on various factors such as the urgency and importance of the performance concern, the time available for analysis, the level of detail of I/O performance information, and the knowledge of jobs and their use of datasets.

There are several approaches that can be employed, such as:

- *Application-driven I/O analysis*. This is used when you have already identified a critical or poorly-performing application and you wish to improve its performance. This can be further extended to I/O analysis of volumes where datasets reside or checking other applications that are using these datasets.
- *Volume-driven analysis*. This is used when a volume has already been identified as an I/O bottleneck. I/O configuration changes may require dataset relocation and an understanding of volume-level performance.
- *Dataset-driven analysis*. This is used when a dataset is critical for application processing or it is causing contention. Dataset-driven analysis is also used when assessing the applicability of techniques to improve performance, such as data-in-memory.
- *Window-level analysis*. This could be part of regular

performance management rather than an activity to solve application window problems. It could also be used as a quick health check to analyse application I/O performance across the window. This type of analysis can be easily performed over multiple windows.

When analysing I/O performance, one should be aware that the granularity and quality of performance data is very important. As a general rule, one should collect enough data to be sure that it is representative (ie too much data will elongate processing and increase the complexity. Insufficient data may cause invalid tuning actions. One will need data for several days to ensure consistency and repeatability). In addition to this rule, please observe the following guidelines when collecting performance data:

- Ensure that critical periods of differing profiles are represented.
- Keep checking that the periods you select are the important ones.
- Be aware that multiple periods can increase the complexity of the analysis.
- Keep collection and periodic analysis going even if things are all right!

Performance data can be summarized for hourly intervals, which is sufficient for performance management. However, your tuning efforts may require much smaller intervals.

ANALYSING I/O COMPONENTS

What I/O indicators should one use? Use contention indicators to analyse performance. These are I/O response times by component, the I/O rate, and cache statistics. One should analyse each I/O component to determine the causes contributing to the response time. To do this one should break down the response time of the selected datasets into components (QUEUE, PEND, CONNECT, DISCONNECT):

- QUEUE time – MVS internal queueing caused by I/O requests waiting for a device while it is busy performing I/O. This response time component can be reduced by improving service time (pend, connect, and disconnect times) or by reducing the rate of I/O requests.
- PEND time – channel and control unit contention, including device busy from another system (if shared DASD).
- CONNECT time – data transfer and search time. Data transfer can be improved by faster channels, and increasing the proportion of I/Os that are cache hits. Searches can be reduced by keeping directories in storage. Short connect time could be an indication of inefficient blocking – increase block sizes. A large connect time, caused by moving large amounts of data, is efficient. Typical examples are DFDSS back-ups and DB2 prefetch. However, long data transfers may delay other jobs using the same volume or channel paths. If the large connect time is caused by long PDS searches, consider options to keep directories in storage or use cacheing.
- DISCONNECT time – seek, latency, and RPS delay. This can be reduced by cacheing, faster DASD, moving datasets, reducing channel and controller busy, and so on. For cached volumes, categorize I/Os into those resolved in cache and those resolved in DASD. Disconnect considerations one should pay attention to are:
  - use cacheing to minimize disconnect time: the disconnect time is zero for a cache hit.
  - reduce seek time by placing high activity datasets close to each other. If the seeking is within one very large dataset, consider splitting or using sequential data striping.
  - applications sharing a volume and using inefficient block sizes may cause excessive seeking (increased seek time and queue time).

– use faster technology to reduce seek time, latency time, and RPS miss time.

– a heavy I/O load to a device and other devices attached to the same controller may cause excessive RPS misses.

COLLECTING I/O DATA AND REPORTING

Where can you find I/O-related information? When enabled by the SMFPRM*xx* TYPE parameter of your system parameter library, DFSMS/MVS creates I/O statistics at the dataset level for DFSMS/MVS-managed datasets. For such datasets behind 3990 DASD storage controls, the statistics are extended to include cacheing information, which is used by Dynamic Cache Management Enhancement to manage cache effectively. The I/O statistics are also recorded in SMF type 42 subtypes 1, 5, and 6 records.

SMF type 42 subtype 1 summarizes, on a storage-class basis, the buffer manager 'hits' (number of page-read requests handled by the buffer manager). A Buffer Manager Facility (BMF) totals section enables analysis of overall BMF performance. There is one storage-class summary section for each storage class. Despite the fact that this summary is not at the dataset level, this information does provide a useful foundation to start to gain some insight into the effectiveness of the dataspace BMF uses. Note: PDSE directory information is cached in a dataspace (SYSBMFDS) while the members are cached in a hiperspace (SYSBMFHS).

SMF type 42 subtype 6 records report EXCPs, I/O performance statistics, and cache statistics at the job and dataset level. This is the best source for application-driven and dataset-driven I/O analysis.

SMF type 42 subtype 5 records give the same data at the storage class level. This data can be used to report I/O performance at the storage class level if your installation has defined several storage classes. This will allow you to determine

whether you are meeting your service levels. These service levels could be formal (negotiated with the users) or informal service targets.

In order to provide a starting point from which one can begin to analyse I/O-related information, I have coded a sample dataset performance report writer. The code is a four-part stream. The first part (DEL42) is a clean-up step that deletes the files to be used in later steps. In the second step (EXT42), SMF records 42 are extracted from the SMF dataset to a file, which can be used as a base of archived records. In the next part (SORT42), previously-extracted records (selection being defined by INCLUDEs condition) are being sorted and copied to a file, which is the input to the DSIO EXEC invoked in the DSIOPERF step.

Several reports are generated by this reporter. The first one is the dataset performance report, consisting of three parts – identification, I/O statistics, and access method statistics for each dataset. I created a variable called weight, which is the product of I/O count (s42dsion) and response time (the mean of response time – s42dsior). This variable is used to indicate as well as to segregate busy datasets.

The next report (storage class performance report) is generated from SMF type 42 subtype 5 records. Interesting to observe is the I/O intensity value. It indicates the degree to which a system is accessing a storage class. Keep in mind that the level of I/O intensity – high or low – is simply a measurement of activity, not necessarily a problem. I use it as an indicator to help show where real problems might exist. I would recommend looking at system-level data during the initial phase, concentrating on I/O intensity because this measurement has an effect on everything else, and looking at the information here can identify a storage class that might be causing multiple levels of problems.

The buffer manager facility performance report is generated from SMF type 42 subtype 1 records. This report shows the

number of PDSE member page reads and PDSE member page read found, by storage class, for the selected period. Read found can occur when a member page is cached in hiperspace by BMF, depending on the direct MSR value in the associated storage class for that PDSE. Only high-performance and commonly-shared PDSE datasets are optionally cached in hiperspace (SYSBMFHS). The absolute number of member page read hits is not meaningful by itself because several factors may cause hit ratios to go both up and down.

Also reported is the number of PDSE directory page reads and PDSE directory page reads found, by storage class, for the selected period. Read found can occur when a directory page is already cached in a data space (SYSBMFDS) by BMF. Directory pages of both types of PDSEs, that is, data and program object PDSEs, are always read into BMF's data space. This data space is created at PDSE support initialization time (IPL time). The data space is mandatory, and all address spaces within an MVS image share it. The absolute number of directory page reads found is not meaningful by itself because several factors may cause hit ratios to go both up and down. The read found ratio can be used to monitor and evaluate DFP changes over time and also to evaluate and test the access behaviour of PDSE candidates.

The interval for producing these records is controlled by a parmlib member (IGDSMS*xx*) specification. The BMFTIME (nnnnn) parameter specifies the number of seconds that SMS is to wait between recording SMF records for Buffer Manager Facility (BMF) cache use. One can specify a value from 1 to 86399 (23 hours, 59 minutes, 59 seconds), and the default is 3600 (one hour). The SMF_TIME keyword, if set to YES, overrides the BMFTIME keyword.

There are two cache parameters that can be used to manage the size of the BMF data space cache. These parameters can be specified in the parmlib member IGWSMS*xx* or in the MVS SETSMS command. One can use these parameters to change the frequency of execution of the BMF LRU and the number

of LRU cycles before buffers are reused for new data. It is likely that most users will continue to let BMF dynamically manage its cacheing. These new options are intended for very heavy users of BMF cacheing, such as Lotus Domino servers.

The LRUTIME(seconds) parameter specifies the amount of time (5 to 60) that BMF will wait between calls to the BMF data space cache LRU (Least Recently Used) routine. The LRU releases inactive buffers in the BMF data space that are used to cache PDSE directory data, HFS directory data, and selected HFS file data (this only applies to HFS in DFSMS 1.4).

The LRUTIME is related to LRUCYCLES. Changes to this parameter will take effect on the next execution of the LRU routine. The default values should be used for most installations. In some very high data rate situations you may want to tune these values.

The LRUCYCLES(cycles) parameter specifies the maximum number of times (5 to 240) that the BMF LRU routine will pass over inactive buffers before making them available for reuse. This parameter sets the maximum value. BMF will dynamically change the number of times it passes over inactive buffers. This report shows the active (smf42buf) and high-water mark (smf42bmx) usage of BMF buffers in the data space. The capacity of the BMF data space is about 570,000 4KB buffers. If the active value frequently reaches 250,000 buffers or the high-water mark reaches 400,000 buffers, one may want to reduce the LRUTIME and LRUCYCLES so buffer data is aged out more quickly.

At present, DFSMS/MVS does not create any specific PDSE dataset-related records showing I/O statistics for a dataset that was handled by Buffer Manager Facility. Therefore I have created a BMF-PDSE combined performance report, which shows Buffer Manager Facility performance for each storage class/interval followed by I/O statistics for PDSE datasets processed during the interval.

It should be noted that this report writer is not comprehensive, but nevertheless it is an open-ended program allowing users to modify and customize the reports generated to meet their installation's needs or requirements. Reporting can be done on both current and historical information provided that an appropriate database was created.

CODE

```
//DEL42     EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=X
//SYSIN     DD *
    DELETE hlq.R42.DATA
    DELETE hlq.SMFCOPY.OUT
    SET MAXCC=Ø
/*
//EXT42    EXEC PGM=IFASMFDP,REGION=5M
//INDA1 DD DSN=your.smf.dataset,DISP=SHR
//OUTDA DD DSN=hlq.SMFCOPY.OUT,DISP=(NEW,PASS),
//         UNIT=SYSDA,SPACE=(CYL,(6Ø,15),RLSE),
//         DCB=(your.smf.dataset)
//SYSPRINT DD SYSOUT=X
//SYSIN DD *
       DATE(yyyyddd,yyyyddd)
       START(Ø9ØØ)
       END(17ØØ)
       INDD(INDA1,OPTIONS(DUMP))
       OUTDD(OUTDA,TYPE(42(1,5,6)))
/*
//SORT42  EXEC PGM=ICETOOL
//TOOLMSG  DD SYSOUT=*
//DFSMSG   DD SYSOUT=*
//RAWSMF   DD DSN=hlq.SMFCOPY.OUT,DISP=SHR
//SMF42    DD DSN=hlq.R42.DATA,
//         SPACE=(CYL,(3Ø,15)),UNIT=SYSDA,DISP=(NEW,KEEP),
//         DCB=(RECFM=VB,LRECL=32756,BLKSIZE=3276Ø)
//TOOLIN   DD *
  SORT FROM(RAWSMF) TO(SMF42) USING(SMF4)
//SMF4CNTL DD *
*
* Eliminate Header and Trailer records
* Sort by date and time
  OPTION SPANINC=RC4,VLSHRT
  INCLUDE COND=(6,1,BI,EQ,42,AND,(23,2,BI,EQ,1,
                OR,23,2,BI,EQ,5,OR,23,2,BI,EQ,6))
  SORT FIELDS=(11,4,PD,A,7,4,BI,A)
```

```
//DSIOPERF EXEC PGM=IKJEFT01,REGION=0M,DYNAMNBR=20
//SYSEXEC  DD DISP=SHR,DSN=your.rexx.lib
//SMF      DD DISP=SHR,DSN=hlq.R42.DATA
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
prof nopref
%DSIO
/*
```

DSIO EXEC

```
/* REXX EXEC to read and format SMF 42 (1,5 & 6) records  */
ADDRESS TSO
numeric digits 12
userid=SYSVAR(SYSUID)
dssr = userid||'.dsn.rpt8'       /* Data set performance report file */
pdsr = userid||'.dsnpds.rpt'     /* PDSE Performance report file     */
dspr = userid||'.dsnbmf.rpt'     /* BMF - PDSE report file           */
bmfr = userid||'.bmf.rpt         /* Buffer Manager Facility file     */
sms  = userid||'.sclass.rpt8'    /* Storage Class performance file   */
x = MSG('OFF')
IF SYSDSN(dssr) = 'OK'
   THEN "DELETE "dssr" PURGE"
IF SYSDSN(pdsr) = 'OK'
   THEN "DELETE "pdsr" PURGE"
IF SYSDSN(dspr) = 'OK'
   THEN "DELETE "dspr" PURGE"
IF SYSDSN(bmfr) = 'OK'
   THEN "DELETE "bmfr" PURGE"
IF SYSDSN(sms) = 'OK'
   THEN "DELETE "sms" PURGE"

"ALLOC FILE(DATASP) DA("dspr")",
   " UNIT(SYSALLDA) NEW TRACKS SPACE(700,99) CATALOG",
   " REUSE RELEASE LRECL(142) RECFM(F B)"

"ALLOC FILE(DSNREP) DA("dssr")",
   " UNIT(SYSALLDA) NEW TRACKS SPACE(600,15) CATALOG",
   " REUSE RELEASE LRECL(316) RECFM(F B)"

"ALLOC FILE(PDSREP) DA("pdsr")",
   " UNIT(SYSALLDA) NEW TRACKS SPACE(300,6) CATALOG",
   " REUSE RELEASE LRECL(160) RECFM(F B)"

"ALLOC FILE(BMFREP) DA("bmfr")",
   " UNIT(SYSALLDA) NEW TRACKS SPACE(300,6) CATALOG",
   " REUSE RELEASE LRECL(70) RECFM(F B)"

 "ALLOC FILE(SMSSC) DA("sms")",
```

```
    " UNIT(SYSALLDA) NEW TRACKS SPACE(300,150) CATALOG",
    " REUSE RELEASE LRECL(155) RECFM(F B)"
w = 1
  rptd.1 = left('Data set performance report',50) left(' ',41,' '),
           left('I/O statistics:',15)             left(' ',119,' '),
           left('Access Method statistics:',25)
  rptd.2 = left(' ',93) left(' I/O',4),
           left(' ',30) left(' ----   Average time (ms)   ----',37),
           left(' ',3,' '),
          left('---- 3990 Control unit cache & I/O statistics ----',52),
           left(' ',3,' ') left('No. blk read',12) left(' ',4,' ') ,
        left('Read I/O dly',16)                 left('Blk written',18) ,
           left('Write I/O dly',15)               left('Dir.',4),
           left('Dir.I/O dly',11)
  rptd.3 = left('Date',11)     left('Time',8)      left('Job  ',8)  ,
           left('DSORG',7)     left('Volser',8)    left('Class',10) ,
           left('blk',4)       left('Data set',28) left('weight',9) ,
           left('#I/Os',5)     left('Resp ',6)     left('Maxresp',7),
           left('Maxserv',8)   left('Conn',7)      left('Pend',6)   ,
           left('Disc',11)     left('IOSQ',6)      left('Ca.C',7)   ,
           left('Ca.h',7)      left('Wr.C',7)      left('Wr.h',8)   ,
           left('Seq',7)       left('RLC',7)       left('ILC',3)    ,
           left('Dev.time',9 ) left('Total',5)     left('Seq',6)    ,
           left('Dir',6)       left('Seq',6)       left('Dir',7)    ,
           left('Total',5)     left('Seq',6)       left('Dir',6)    ,
           left('Seq',6)       left('Dir',6)       left('reads',6)  ,
           left('read',4)      left('write',5)
  rptd.4 = left('-',315,'-')
  "EXECIO * DISKW DSNREP (STEM rptd.)"
  wrr.1 =  left(' ',47,' ')  left('#Blocks',11)  left('#Dir.',5),
           left(' ',13,' ')  left('Max',3)       left(' ',9,' '),
           left('I/O',15)    left('  -- I/O Delays --',20)
  wrr.2 =  left(' ',4,' ')   left('Data set',23) left('Class',10),
           left('#I/Os',5)   left('Read',5)      left('Write',6) ,
           left('read/wr.',8) ,
           left('Resp ',8)   left('resp',7)      left('serv',7),
           left('weight',11) left('Seq.R/W',12),
           left('Dir.R/W',14)                    left('DIR r/w',7)
  wrr.3  = left(' ',4,' ')   left('-',136,'-')

  pww.1 =  left('PDSE Performance report',50)
  pww.2 =  left(' ',1)
  pww.3 =  left(' ',71,' ')   left('#Blocks',11) left('#Dir.',5),
           left(' ',13,' ')   left('Max',3)      left(' ',13,' '),
           left('  -- I/O Delays --',20)
  pww.4  = left('Date',11)    left('Time',8)     left('Job  ',8) ,
           left('Data set',22)                   left('Class',10),
           left('#I/Os',5)    left('Read',5)     left('Write',6) ,
           left('read/wr.',8) left('Resp ',8),
           left('resp',7)     left('serv',9)     left('Seq.R/W',12),
```

```
          left('Dir.R/W',14)                    left('DIR r/w',7)
  pww.5  = left('-',155,'-')
  "EXECIO * DISKW PDSREP (STEM pww.)"

  bbf.1  = left('Buffer Manager Facility performance report',5Ø)
  bbf.2  = left(' ',1)
  bww.1  = left('BMF - PDSE combined preformance report',5Ø)
  bww.2  = left(' ',1)
 "EXECIO * DISKW BMFREP (STEM bbf.)"
 "EXECIO * DISKW DATASP (STEM bww.)"
 'EXECIO * DISKR SMF ( STEM x. FINIS'
    do i = 1 to x.Ø
/*-----------------------------------------------------------------*/
/*    Header for SMF record type 42                                */
/*-----------------------------------------------------------------*/
 smf42rty  = c2d(SUBSTR(x.i,2,1))     /*         SMF record type */
IF smf42rty= '42'  Then do
 smf42tme = smf(c2d(SUBSTR(x.i,3,4)))         /* Decode SMF time */
 smf42dte = SUBSTR(c2x(SUBSTR(x.i,7,4)),3,5)  /* Unpack SMF date */
 smd42sid = SUBSTR(x.i,11,4)            /* System identification */
 smf42ssi = SUBSTR(x.i,15,4)               /*       Subsystem id */
 smf42sty = c2d(SUBSTR(x.i,19,2))          /*     Record subtype */
 smf42nt  = c2d(SUBSTR(x.i,21,2))          /* Number of triplets */
/*-----------------------------------------------------------------*/
/*    Product Section                                              */
/*-----------------------------------------------------------------*/
Select
  when smf42sty =  1  Then call subt1
  when smf42sty =  5  Then call subt5
  otherwise do                        /*Proccess subtype 6        */
  smf42ops = c2d(SUBSTR(x.i,25,4))   /*Offset to product section */
  smf42lps = c2d(SUBSTR(x.i,29,4))   /*Lenght to product section */
  smf42nps = c2d(SUBSTR(x.i,31,4))   /*Number to product sections*/
 IF smf42ops <> Ø AND smf42lps <> Ø Then do
    smf42ops=smf42ops -3
    smf42pdl = SUBSTR(x.i,smf42ops,8)          /*   Product level*/
    smf42pdn = SUBSTR(x.i,smf42ops+8,1Ø)       /*    Product name*/
    smf42psv = c2d(SUBSTR(x.i,smf42ops+18,1))  /*Subtype ver. no.*/
      SELECT
        when smf42psv='Ø' then vhead ='No vol.header section'
        when smf42psv='1' then vhead ='Vol. header exists'
      END
  end
/*-----------------------------------------------------------------*/
/*   SMF42 subtype 6 job header section                            */
/*-----------------------------------------------------------------*/
   smf42jho = c2d(SUBSTR(x.i,33,4))      /*Offset to job section */
   smf42jhl = c2d(SUBSTR(x.i,37,2))      /*Length of job section */
   smf42jhn = c2d(SUBSTR(x.i,39,2))      /*Number of job sections*/
/*-----------------------------------------------------------------*/
```

```
/*     Job header section                                          */
/*-----------------------------------------------------------------*/
IF smf42jho <> Ø  Then do
 smf42jho=smf42jho -3
 s42jdjnm = SUBSTR(x.i,smf42jho,8)                    /*  Job name  */
 s42jdrst = smf(c2d(SUBSTR(x.i,smf42jho+8,4)))        /*Start time  */
 s42jdrsd = SUBSTR(c2x(SUBSTR(x.i,smf42jho+12,4)),3,5)  /*Start date  */
 s42jduid = SUBSTR(x.i,smf42jho+16,8)                 /*  User id.  */
 s42jddso = c2d(SUBSTR(x.i,smf42jho+24,4))    /*Offset to 1st.ds sec. */
 s42jddsl = c2d(SUBSTR(x.i,smf42jho+28,2))    /*Length of DS header   */
 s42jdcod = c2d(SUBSTR(x.i,smf42jho+3Ø,1))    /*Record close type     */
 s42jdpgn = c2d(SUBSTR(x.i,smf42jho+32,2))    /*Job perf. group num.  */
 s42jdiol = c2d(SUBSTR(x.i,smf42jho+34,2))   /*Len.of ds IO STAT sec. */
 s42jdaml = c2d(SUBSTR(x.i,smf42jho+36,2))   /*Len of ds AMS STAT sec.*/
 s42jdwsc = SUBSTR(x.i,smf42jho+44,8)        /*WLM: Service Class Name*/
 s42jdwld = SUBSTR(x.i,smf42jho+52,8)        /*WLM: Workload name     */
 end
/*-----------------------------------------------------------------*/
/*     DATA SET hdr sect                                           */
/*-----------------------------------------------------------------*/
 Select
   when s42jddso > Ø then s42dsx= s42jddso - 3 /* FIRST ds offset*/
   otherwise nop
 End
/*-----------------------------------------------------------------*/
/*    Get data for the first dataset                               */
/*-----------------------------------------------------------------*/
 Select
  when s42dsx > Ø then  call DATASET s42dsx
  otherwise nop
 End
/*-----------------------------------------------------------------*/
/*    Print data for the first dataset                             */
/*-----------------------------------------------------------------*/
 dd.1 = left(date('n',smf42dte,'j'),11)  smf42tme,
        left(s42jdjnm,8)       left(dsnn,7),
        left(s42dsvol,8)       left(s42dssc,8),
        right(s42dsbsz,6)      left(s42dsn,25) right(weight,9)
  rr.1=dd.1||ior.1||amr.1
  "EXECIO * DISKW DSNREP (STEM rr.)"
/*-----------------------------------------------------------------*/
/*    Get data for each next dataset in the dataset chain          */
/*-----------------------------------------------------------------*/
 Do while s42dsnxt   <> Ø
    s42dsq =  s42dsnxt -3
    call DATASET s42dsq
/*-----------------------------------------------------------------*/
/*    Print data for each next dataset in the dataset chain        */
/*-----------------------------------------------------------------*/
 sd.1 = left(' ',2Ø,' ')   left(s42jdjnm,8),
```

```
        left(dsnn,7)       left(s42dsvol,8),
        left(s42dssc,8)    right(s42dsbsz,6) left(s42dsn,25) ,
        right(weight,9)
 srr.1=sd.1||ior.1||amr.1
 "EXECIO * DISKW DSNREP (STEM srr.)"
       end
     end
  end
end
end
/*-------------------------------------------------------------*/
/*   Close and free allocated files                            */
/*-------------------------------------------------------------*/
 "EXECIO Ø DISKW DSNREP (FINIS "
 "EXECIO Ø DISKW PDSREP (FINIS "
 "EXECIO Ø DISKW DATASP (FINIS "
 "EXECIO Ø DISKW BMFREP (FINIS "
 "EXECIO Ø DISKW SMSSC  (FINIS "
 say
 say 'Data set performance report file: 'dssr
 say 'PDSE Performance report file    : 'pdsr
 say 'Buffer Manager Facility  file   : 'bmfr
 say 'BMF - PDSE report file          : 'dspr
 say 'Storage Class performance file  : 'sms
 "FREE FILE(SMF DSNREP PDSREP DATASP BMFREP SMSSC)"
exit

DATASET:
/*-------------------------------------------------------------*/
/* Dataset header (detail) section                             */
/*-------------------------------------------------------------*/
parse arg offds
 s42dsnxt   =     SUBSTR(x.i,offds,4)       /* Offset to next DS hdr */
Select
when s42dsnxt ='4Ø4Ø4Ø4Ø'X  then s42dsnxt = Ø
otherwise s42dsnxt=c2d(SUBSTR(x.i,offds,4)) /* Offset to next DS hdr */
End
 s42dsn    = SUBSTR(x.i,offds+4,44)               /*  Dataset name */
 s42dstyp  = c2d(SUBSTR(x.i,offds+48,1))          /*  Dataset type */
        SELECT
          when s42dstyp ='1' then dsnn ='PS      '
          when s42dstyp ='2' then dsnn ='PDS     '
          when s42dstyp ='3' then dsnn ='PDSE    '
          when s42dstyp ='4' then dsnn ='DA      '
          when s42dstyp ='5' then dsnn ='IS      '
          when s42dstyp ='6' then dsnn ='EXCP    '
          when s42dstyp ='7' then dsnn ='Ext.    '
          when s42dstyp ='1Ø' then dsnn ='HFS    '
          when s42dstyp ='16' then dsnn ='KSDSd  '
          when s42dstyp ='17' then dsnn ='KSDSi  '
```

```
          when s42dstyp ='18' then dsnn ='vRRDSd '
          when s42dstyp ='19' then dsnn ='vRRDSi '
          when s42dstyp ='2Ø' then dsnn ='fRRDS  '
          when s42dstyp ='21' then dsnn ='LDS    '
          when s42dstyp ='22' then dsnn ='ESDS   '
          otherwise nop
       END
 s42dscod= c2d(SUBSTR(x.i,offds+49,1))       /*Entry descriptor flags */
 s42dsfl1  = c2d(SUBSTR(x.i,offds+5Ø,1))
 s42dsfl2  =x2b(c2x(SUBSTR(x.i,offds+5Ø,1)))       /*  DS descr.flags */
Select
  when s42dsfl2 = '00000000' then flag='NSR       '
  when s42dsfl2 = '00000001' then flag='Compressed'
  when s42dsfl2 = '00000010' then flag='Extended format'
  when s42dsfl2 = '00000100' then flag='Program lib.'
  when s42dsfl2 = '00001000' then flag='Non-VSAM FB rec.'
  when s42dsfl2 = '00001100' then flag='FB program lib'
  when s42dsfl2 = '00010000' then flag='Open for EXCP proc'
  when s42dsfl2 = '00010100' then flag='Excp proces. prog.lib'
  when s42dsfl2 = '00011000' then flag='Excp proces. fb rec'
  when s42dsfl2 = '10000000' then flag='VSAM buffer flags'
  when s42dsfl2 = '10000010' then flag='VSAM extended format'
  otherwise  flag='??????'
end
 s42dsioo  = c2d(SUBSTR(x.i,offds+52,4))          /* Offset to IO sec */
 s42dsamo  = c2d(SUBSTR(x.i,offds+56,4))          /* Offset to AMS sec*/
 s42dsvol  = SUBSTR(x.i,offds+6Ø,6)               /* Volume serial    */
 s42dsdev  = c2d(SUBSTR(x.i,offds+66,2))          /* Device number    */
 s42dssc   = SUBSTR(x.i,offds+68,8)               /*Storage class name*/
 s42dsbsz  = c2d(SUBSTR(x.i,offds+76,4))          /* Block size       */
 s42dstrp  = c2d(SUBSTR(x.i,offds+8Ø,2))          /*Number of stripes */
/*-----------------------------------------------------------------*/
/*  Call to process I/O statistics                                 */
/*-----------------------------------------------------------------*/
 Select
   when s42dsioo > Ø then call IOSEC  s42dsioo
   otherwise do
   ior.1=left('     no I/O data available',127)
   weight = Ø
   end
 End
/*-----------------------------------------------------------------*/
/*  Call to process AM statistics                                  */
/*-----------------------------------------------------------------*/
 Select
   when s42dsamo > Ø then call AMS  s42dsamo
   otherwise amr.1=left('     no AMS data available',3Ø)
 End
/*-----------------------------------------------------------------*/
/* PDSE report                                                     */
```

```
/*-------------------------------------------------------------*/
Select
 when  dsnn = pdse & s42dssc  >'        ' then do
  pde.1 = left(date('n',smf42dte,'j'),11)  smf42tme,
          left(s42jdjnm,8)   left(s42dsn,22),
          left(s42dssc,8)    right(s42dsion,5),
          right(totalr,5)    right(totalw,5),
          right(s42amzrb,4)  right(s42amzwr,4),
          right(s42dsior,7)  right(s42dsmxr,7),
          right(s42dsmxs,7)  right(s42amsrr,6),
          right(s42amswr,6)  right(s42amdrr,6),
          right(s42amdwr,6)  right(s42amzrr,6),
          right(s42amzwr,6)
    "EXECIO * DISKW PDSREP (STEM pde.)"
  pds.w = right(w,4,' ') left(s42dsn,23)   left(s42dssc,8),
          right(s42dsion,5),
          right(totalr,5)    right(totalw,5),
          right(s42amzrb,4)  right(s42amzwr,4),
          right(s42dsior,7)  right(s42dsmxr,7),
          right(s42dsmxs,7)  right(weight,9)  ,
          right(s42amsrr,6)  right(s42amswr,6),
          right(s42amdrr,6)  right(s42amdwr,6),
          right(s42amzrr,6)  right(s42amzwr,6)
w = w + 1
 end
otherwise nop
End
return

IOSEC:
/*-------------------------------------------------------------*/
/*     Dataset I/O statistics:                                 */
/*     I/O response/service time components are recorded       */
/*     in multiples of 128 micro-seconds.                      */
/*     The length of these statistics must be identical        */
/*     with the length of DSSBSTAT in the DSSB.                */
/*-------------------------------------------------------------*/
parse arg offio
           sioo= offio -3
Select
  when sioo > Ø & sioo < 9999 then do
 weight = Ø
 s42dsior = (c2d(SUBSTR(x.i,sioo,4)))*128E-3      /*Response time    */
 s42dsioc = (c2d(SUBSTR(x.i,sioo+4,4)))*128E-3    /*Avg I/O connect  */
 s42dsiop = (c2d(SUBSTR(x.i,sioo+8,4)))*128E-3    /*Avg I/O pending  */
 s42dsiod = (c2d(SUBSTR(x.i,sioo+12,4)))*128E-3   /*Avg I/O disconnect*/
 s42dsioq = (c2d(SUBSTR(x.i,sioo+16,4)))*128E-3  /*Avg cntl unit queue*/
 s42dsion =  c2d(SUBSTR(x.i,sioo+2Ø,4))         /*Total number of I/Os*/
 weight   = s42dsion* s42dsior                    /*DS I/O weight: not*/
                                                  /*printed          */
```

```
/*-----------------------------------------------------------------*/
/*    399Ø Control unit cache statistics                           */
/*-----------------------------------------------------------------*/
 s42dscnd =  c2d(SUBSTR(x.i,sioo+24,4))         /*# cache candidates  */
 s42dshts =  c2d(SUBSTR(x.i,sioo+28,4))         /*# of cache hits     */
 s42dswcn =  c2d(SUBSTR(x.i,sioo+32,4))         /*# of write cand.    */
 s42dswhi =  c2d(SUBSTR(x.i,sioo+36,4))         /*# of write hits     */
 s42dsseq =  c2d(SUBSTR(x.i,sioo+4Ø,4))         /*# of sequential I/Os*/
 s42dsrlc =  c2d(SUBSTR(x.i,sioo+44,4))          /*# of RLC I/Os      */
 s42dsicl =  c2d(SUBSTR(x.i,sioo+48,4))          /*# of ILC I/Os      */
 s42dsdaØ = (c2d(SUBSTR(x.i,sioo+52,4)))*128E-3   /*Average I/O device*/
                                                  /* active-only time */
/*-----------------------------------------------------------------*/
/*    Following two fields are not part of DSSSBSTAT               */
/*-----------------------------------------------------------------*/
 s42dsmxr = (c2d(SUBSTR(x.i,sioo+56,4)))*128E-3
                                          /*Max. ds I/O response time*/
 s42dsmxs = (c2d(SUBSTR(x.i,sioo+6Ø,4)))*128E-3
                                          /*Max. ds service time     */
 iosqueque=s42dsmxr-s42dsmxs
 ior.1 = right(s42dsion,7)    right(s42dsior,7),
         right(s42dsmxr,7)    right(s42dsmxs,7),
         right(s42dsioc,7)    right(s42dsiop,7),
         right(s42dsiod,7)    right(s42dsioq,7),
         right(s42dscnd,7)    right(s42dshts,7),
         right(s42dswcn,7)    right(s42dswhi,7),
         right(s42dsseq,7)    right(s42dsrlc,7),
         right(s42dsicl,7)    right(s42dsdaØ,7)
   end
 otherwise nop
End
return

AMS:
/*-----------------------------------------------------------------*/
/*     Dataset AMS section                                         */
/*-----------------------------------------------------------------*/
parse arg offam
           samo= offam -3
Select
  when samo > Ø & samo < 9999 then do
 s42amsrb = c2d(SUBSTR(x.i,samo,4))         /*Seq. read no. of blk    */
 s42amsrr =(c2d(SUBSTR(x.i,samo+4,4)))*128E-3   /*Seq. read I/O dly   */
 s42amswb = c2d(SUBSTR(x.i,samo+8,4))        /*Seq. write no. of blk  */
 s42amswr =(c2d(SUBSTR(x.i,samo+12,4)))*128E-3  /*Seq. write I/O dly  */
 s42amdrb = c2d(SUBSTR(x.i,samo+16,4))          /*Dir. read # blk     */
 s42amdrr =(c2d(SUBSTR(x.i,samo+2Ø,4)))*128E-3
                                          /*Dir. read I/O total dly */
 s42amdwb = c2d(SUBSTR(x.i,samo+24,4))      /*Dir. write no. of blk   */
 s42amdwr =(c2d(SUBSTR(x.i,samo+28,4)))*128E-3
```

```
                                          /*Dir. write total I/O dly*/
 s42amzrb = c2d(SUBSTR(x.i,samo+32,4))     /*No. of directory reads  */
 s42amzrr =(c2d(SUBSTR(x.i,samo+36,4)))*128E-3
                                         /*Directory read I/O dly  */
 s42amzwb = c2d(SUBSTR(x.i,samo+40,4))     /*No. of directory writes */
 s42amzwr =(c2d(SUBSTR(x.i,samo+44,4)))*128E-3
                                         /*Directory write I/O dly */
 totalr = s42amsrb + s42amdrb            /*Read total= seq.r+dir.r */
 totalw = s42amswb + s42amdwb            /*Write tot = seq.w+dir.w */
 totals = s42amsrb + s42amswr            /*Seq tot=read.s+write.s  */
 totald = s42amdrb + s42amdwb            /*Dir tot=read.d+write.d  */
 amr.1 = right(totalr,6)    right(s42amsrb,6),
         right(s42amdrb,6)  right(s42amsrr,6),
         right(s42amdrr,6)  right(totalw,6),
         right(s42amswb,6)  right(s42amdwb,6),
         right(s42amswr,6)  right(s42amdwr,6),
         right(s42amzrb,6)  right(s42amzrr,6) right(s42amzwr,6)
   end
 otherwise nop
End
return

SUBT1:
/*-------------------------------------------------------------------*/
/* REXX EXEC to read and format SMF 42.1 records (BMF)               */
/* Subtype 1 summarizes, on a storage-class basis, the buffer        */
/* manager 'hits' (number of page-read requests handled by the       */
/* buffer manager). A Buffer Manager Facility (BMF) totals section*/
/* (64 bytes) enables analysis of overall BMF performance. There     */
/* is one storage-class summary section (64 bytes) for each          */
/* storage class                                                     */
/*-------------------------------------------------------------------*/
 hd.1 = left('Date and time',20) left('4K BMF',9),
        left("hit %",6)          left('Mem.r',5) ,
        left('Mem.f',5)          left('Dir.r',5) ,
        left('Dir.f',5)          left('bmfhwm',6)
 hd.2 = left('-',70,'-')
"EXECIO * DISKW DATASP (STEM hd.)"
"EXECIO * DISKW BMFREP (STEM hd.)"
   smf42bmo = c2d(SUBSTR(x.i,33,4))  /* Offset to BMF totals section  */
   smf42bml = c2d(SUBSTR(x.i,37,2))  /* Length of BMF totals section  */
   smf42bmn = c2d(SUBSTR(x.i,39,2))  /* Number of BMF totals sections */
   smf42sco = c2d(SUBSTR(x.i,41,4))  /*Offset to stg.class summary sec*/
   smf42scl = c2d(SUBSTR(x.i,45,2)) /* Length of stg.class summary sec*/
   smf42scn = c2d(SUBSTR(x.i,47,2)) /* Number of stg.class summary sec*/
/*-------------------------------------------------------------------*/
/*   SMF42 subtype 1 header section (BMF Statistics)                 */
/*-------------------------------------------------------------------*/
   IF (smf42bmn > 0) Then do
    do pp = 0 to (smf42bmn -1)
```

```
    bmo  = (smf42bmo + (pp*smf42bml))- 3
smf42tna = c2d(SUBSTR(x.i,bmo,4))        /* Total no. of stg.classes  */
smf42tmt = c2d(SUBSTR(x.i,bmo+4,4))       /* Interval length         */
smf42trt = c2d(SUBSTR(x.i,bmo+8,4))       /* member data page reads  */
smf42trh = c2d(SUBSTR(x.i,bmo+12,4))      /* found in BMF            */
smf42tdt = c2d(SUBSTR(x.i,bmo+16,4))      /* dir.data page reads     */
smf42tdh = c2d(SUBSTR(x.i,bmo+20,4))      /* found in BMF            */
  Select
   when smf42tdt > 0 then  rhit = (smf42tdh/smf42tdt)*100
   otherwise               rhit= '000000'
  END
smf42buf = c2d(SUBSTR(x.i,bmo+24,4))       /* active BMF 4K buffers   */
smf42bmx = c2d(SUBSTR(x.i,bmo+28,4))       /* High-water mark of BMF  */
smf42lru = c2d(SUBSTR(x.i,bmo+32,2))       /* BMF LRU interval time   */
smf42uic = c2d(SUBSTR(x.i,bmo+34,2))       /* BMF LRU cycles          */
/*Select
 when rhit > 0 then do */
 bm.1 = left(date('n',smf42dte,'j'),11)  smf42tme ,
       right(smf42buf,5)     format(rhit,6,3) ,
       right(smf42trt,4)     right(smf42trh,4),
       right(smf42tdt,6)     right(smf42tdh,6),
       right(smf42bmx,5)
"EXECIO * DISKW DATASP (STEM bm.)"
"EXECIO * DISKW BMFREP (STEM bm.)"
/*  end
otherwise nop
End */
   IF (smf42scn > 0) Then do
    do ss = 0 to (smf42scn -1)
    sco  = (smf42sco + (ss*smf42scl))- 3
smf42pnl = c2d(SUBSTR(x.i,sco,2))          /*Storage class name length*/
smf42pnn =     SUBSTR(x.i,sco+2,30)        /* Storage class name      */
smf42srt = c2d(SUBSTR(x.i,sco+32,4))       /* member data page reads  */
smf42srh = c2d(SUBSTR(x.i,sco+36,4))       /* found in BMF            */
smf42sdt = c2d(SUBSTR(x.i,sco+40,4))       /* dir.data page reads     */
smf42sdh = c2d(SUBSTR(x.i,sco+44,4))       /* found in BMF            */
 Select
   when smf42sdt > 0 then  rhi  = (smf42sdh/smf42sdt)*100
   otherwise               rhi = '000000'
 End
/*Select
 when rhi  > 0 then do */
 bf.1 = left(' ',2,' ')    left('Storage class: ',14) ,
        left(smf42pnn,8)   format(rhi,6,3) ,
        right(smf42srt,4)  right(smf42srh,4),
        right(smf42sdt,6)  right(smf42sdh,6)
"EXECIO * DISKW DATASP (STEM bf.)"
"EXECIO * DISKW BMFREP (STEM bf.)"
/* end
otherwise nop
```

```
End */
 END
numm = w -1
comm.1= left(' ',2,' ')
Select
 when numm > Ø then do
 comm.2= left('Number of PDSE files in this interval: ',4Ø),
         left(numm,8)
 end
 otherwise comm.2= left('No PDSE files processed in this interval',45)
End
comm.3= left(' ',2,' ')
com.1 = left(' ',2,' ')
"EXECIO * DISKW DATASP (STEM comm.)"
 Select
  when numm > Ø then do
"EXECIO * DISKW DATASP (STEM wrr.)"
  end
 otherwise nop
 End
"EXECIO * DISKW DATASP (STEM pds.)"
"EXECIO * DISKW DATASP (STEM com.)"
drop pds.
w = 1
bl.1 = left(' ',1)
"EXECIO * DISKW BMFREP (STEM bl.)"
return

SUBT5:
a= 1
/*-----------------------------------------------------------------*/
/* REXX EXEC to read and format SMF 42.5 records                   */
/*                                                                 */
/* Header for Storage Class performance report                     */
/*-----------------------------------------------------------------*/
  schd.1 = left('Storage Class performance report',5Ø)
  schd.2 = left(' ',47) left(' ----   Average time (ms)   ----',37),
           left(' ',19,' '),
           left('- 399Ø Control unit cache & I/O statistics --',44)
  schd.3 = left('Date',11)   left('Time',1Ø)   left('Class',9) ,
           left('# I/Os',9)  left('Resp ',9)   left('Conn',9)  ,
           left('Pend',9)    left('Disc',8)    left('IOSQ',4)  ,
           left('IO Rate',7) left('Intens.',7)                 ,
           left('Ca.C',4)    left('Ca.h',4)    right('Hit%',5) ,
           right('Wr.C',5)   right('Wr.h',5)   right('Seq',4)  ,
           right('RLC',4)    right('ILC',3)    left('Dev.time',9 )
  schd.4 = left('-',15Ø,'-')
/*-----------------------------------------------------------------*/
/*    Header for SMF record type 42.5 (storage class portion only)   */
/*-----------------------------------------------------------------*/
```

```
   smf42sro = c2d(SUBSTR(x.i,33,4))  /* Offset to SC response time sec*/
   smf42srl = c2d(SUBSTR(x.i,37,2))  /* Length of SC response time sec*/
   smf42srn = c2d(SUBSTR(x.i,39,2))  /* Number of SC response time sec*/
/*-------------------------------------------------------------------*/
/*   Storage Class Response Time Section (SMF42 subtype 5)           */
/*   I/O response and service time components are recorded in        */
/*   multiples of 128 micro-seconds. Converted to milliseconds.      */
/*-------------------------------------------------------------------*/
smssb.1 = left(' ',3)
   IF (smf42srn > 0) Then do
    do pp = 0 to (smf42srn -1)
    sro  = (smf42sro + (pp*smf42srl))- 3
s42scrnl = c2d(SUBSTR(x.i,sro,2))        /* Storage class name length */
s42scrnn =     SUBSTR(x.i,sro+2,30)      /* Storage class name      */
s42scior = c2d(SUBSTR(x.i,sro+32,4))     /* Response time           */
    cior = s42scior*128E-3               /* Converted to millisecond*/
s42scioc = c2d(SUBSTR(x.i,sro+36,4))     /* Avg I/O connect time    */
    cioc = s42scioc*128E-3               /* Converted to millisecond*/
s42sciop = c2d(SUBSTR(x.i,sro+40,4))     /* Avg I/O pending time    */
    ciop = s42sciop*128E-3               /* Converted to millisecond*/
s42sciod = c2d(SUBSTR(x.i,sro+44,4))     /* Avg I/O disconnect time */
    ciod = s42sciod*128E-3               /* Converted to millisecond*/
s42scioq = c2d(SUBSTR(x.i,sro+48,4))     /* Avg cntl unit queue time*/
    cioq = s42scioq*128E-3               /* Converted to millisecond*/
s42scion = c2d(SUBSTR(x.i,sro+52,4))     /* Total number of I/Os    */
iorate = format(s42scion/18000,8,4)
ioint  = format(iorate*cior,8,4)
/*-------------------------------------------------------------------*/
/*       3990 Control unit cache statistics                          */
/*-------------------------------------------------------------------*/
s42sccnd = c2d(SUBSTR(x.i,sro+56,4))       /* No. of cache candidates */
s42schit = c2d(SUBSTR(x.i,sro+60,4))       /* No. of cache hits       */
s42scwcn = c2d(SUBSTR(x.i,sro+64,4))       /* No. of write candidates */
s42scwhi = c2d(SUBSTR(x.i,sro+68,4))       /* No. of write hits       */
s42scseq = c2d(SUBSTR(x.i,sro+72,4))       /* No. of sequential I/Os  */
s42scrlc = c2d(SUBSTR(x.i,sro+76,4))       /* No. of record level     */
                                           /* cache I/O operations:RLC*/
s42scicl = c2d(SUBSTR(x.i,sro+80,4))       /* No.of inhibit cache     */
                                           /* load I/O operations :ILC*/
s42scdao = c2d(SUBSTR(x.i,sro+84,4))
                                    /* Avg I/O device-active-only time*/
    cdao = s42scdao*128E-3                 /* Converted to millisecond*/
Select
 when s42sccnd > 0 then  rhit = (s42schit/s42sccnd)*100
 otherwise               rhit= '000000'
END
/*-------------------------------------------------------------------*/
/*   Printed Storage Class performance variables:                    */
/*-------------------------------------------------------------------*/
smssc.a = left(date('n',smf42dte,'j'),11) left(smf42tme,10) ,
```

```
          left(s42scrnn,8) ,                  /*   Storage class name */
          right(s42scion,6) ,                 /* Total number of I/Os */
          right(cior,9) ,                     /*   Response time (ms) */
          right(cioc,9),              /*     Avg I/O connect time (ms) */
          right(ciop,9),              /*     Avg I/O pending time (ms) */
          right(ciod,9),              /*  Avg I/O disconnect time (ms) */
          right(cioq,6),              /* Avg cntl unit queue time (ms) */
          right(iorate,7,4),
          right(ioint,7,4),
          right(s42sccnd,4),                  /*     Cache candidates */
          right(s42schit,4),                  /*           Cache hits */
          format(rhit,3,2),                   /*      Cache hit ratio */
          right(s42scwcn,4),                  /*     Write candidates */
          right(s42scwhi,6),                  /*           Write hits */
          right(s42scseq,4),                  /*      Sequential I/Os */
          right(s42scrlc,3),                  /*             RLC I/Os */
          right(s42scicl,3),                  /*             ILC I/Os */
          right(cdao,8)               /*Avg I/O device-active-only(ms) */
 a = a +1
end
 "EXECIO * DISKW SMSSC (STEM schd.)"
 "EXECIO * DISKW SMSSC (STEM smssc.)"
 "EXECIO * DISKW SMSSC (STEM smssb.)"
  end
  drop smssc.
return

SMF: procedure
/* REXX - convert a SMF time  */
arg time
    time1    = time % 100
    hh       = time1 % 3600
    hh       = RIGHT("0"||hh,2)
    mm       = (time1 % 60) - (hh * 60)
    mm       = RIGHT("0"||mm,2)
    ss       = time1 - (hh * 3600) - (mm * 60)
    ss       = RIGHT("0"||ss,2)
    otime  = hh||":"||mm||":"||ss             /* Compose SMF time*/
    return otime
```

Mile Pekic
Systems Programmer (Serbia and Montenegro)

Repairing a full VTOC index in an MVS DASD subsystem

The following discusses the steps required to repair a VTOC index when it is reported as being full.

When creating DASD volumes with indexed VTOCs you should attempt to size the index accordingly for the number of datasets that will be positioned on the volume in question.

The BUILDIX command, a part of ICKDSF, can be used to generate indexed VTOCs and also to convert volumes from indexed VTOCs back to OSVTOCs. The MVS version of ICKDSF checks the general resource profile for a facility class profile of STGADMIN.ICK.BUILDIX. If RACF is not installed or the facility STGADMIN.ICK.BUILDIX is not created, the BUILDIX command executes with no authorization check. This is inadvisable and security should be performed to allow only authorized users to execute this command.

The format of the BUILDIX command is shown below:

```
BUILDIX
   DDNAME(dname)
   IXVTOC|OSVTOC
  PURGE|NOPURGE
```

where:

- DDNAME – identifies the volume to be operated on (a required parameter). The volume being operated on must be ONLINE.
- IXVTOC – converts OSVTOC to an indexed VTOC. It is mutually exclusive with OSVTOC.
- OSVTOC – converts a volume in IXVTOC format to OSVTOC. It is mutually exclusive with IXVTOC.
- PURGE – deletes the indexed VTOC dataset when converting from IXVTOC format to OSVTOC format. This

is an optional parameter and is mutually exclusive with NOPURGE.

- NOPURGE – retains the SYS1.VTOCIX dataset when converting to OSVTOC format. This is an optional parameter that is mutually exclusive with the PURGE parameter.

Before you invoke BUILDIX to change an indexed VTOC, you must pre-allocate the index dataset in a separate job or job step, or you must provide a DD card describing the index dataset in the same job step so the scheduler allocates the index before the command runs. The name of the index dataset should always begin with SYS1.VTOCIX. IBM recommends that each index dataset in your installation have a unique third-level qualifier. The recommended convention is SYS1.VTOCIX.volser. This prevents ENQ lockouts on all other volumes needing IXVTOC services. If the first character of the volser is numeric, use another convention. The convention used for the INDEX parameter of the INIT command is to replace the first character of the volser with the letter V and the last five characters of all volsers starting with a numeric character must be unique. The name SYS1.VTOCIX is a reserved name in systems supporting the indexed VTOC, and only one dataset per volume can begin with this prefix. The index must exist in a contiguous area of space on DASD. This can determine how you recover from a full condition. I will come to that later.

In the event of the VTOC index becoming full you will probably receive a message indicating this. The most common is the MSGIEC614I RC08 message. You can also receive:

```
IGD17273I:
ALLOCATION HAS FAILED FOR ALL VOLUMES SELECTED FOR DATA SET
```

At this point you should attempt to stop further allocations to the volume. If you have SMS, DISABLE the volume in SMS. If not, mount the volume as PRIVATE.

You can then decide on the method of recovery. First of all you

need to decide how big you want to make the VTOC index. There is no hard and fast rule, but do try to allocate enough space to handle all your datasets. Appendix C of the ICKDSF manual gives some advice on size calculations. I have found that on 3390 model 3s a VTOC index of 15 tracks is normally sufficient. On Model 9 3390s, I usually allocate a VTOC index of 30 tracks or 45 tracks. On Model 27s, I allocate 150 tracks. You need to determine whether you can obtain enough contiguous storage to allocate the VTOC index next to your VTOC. If you cannot, the chances are you will need to move all data off the volume, and then just re-initialize it with a larger VTOC index. If you can get enough contiguous space or move small amounts of data to get it, you can follow the quick recovery method.

First ensure that the volume is mounted as PRIVATE. Then run the JCL below:

```
//JXB7884A  JOB   (7884),CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1)
//STEP1     EXEC  PGM=ICKDSF,PARM='NOREPLYU'
//SYSPRINT  DD    SYSOUT=A
//DDCARD    DD    UNIT=(3390,,DEFER),VOL=(PRIVATE,SER=PRD001),
//                DISP=OLD
//SYSIN     DD        *
 BUILDIX  DDNAME(DDCARD)  OS  PURGE
/*
```

This converts the VTOC to OS format and deletes the existing VTOC index dataset. All datasets on the volume will still be accessible. By coding PARM='NOREPLYU' you remove the need for the operators to reply U to message ICK508A on their consoles.

Once purged, you need to re-allocate the new index. This can be done as part of the conversion job. The JCL below allocates a new index that will be contiguous in its space allocation and 30 tracks in length:

```
//JXB7884A  JOB   (7884),CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1)
//STEP1     EXEC  PGM=ICKDSF,PARM='NOREPLYU'
//SYSPRINT  DD    SYSOUT=A
//DDCARD    DD    UNIT=3390,VOL=(PRIVATE,SER=PRD001),
//                DISP=(NEW,KEEP,DELETE),SPACE=(TRK,30,,CONTIG),
```

```
//                DSN=SYS1.VTOCIX. PRDØØ1
//SYSIN      DD        *
 BUILDIX  DDNAME(DDCARD)  IXVTOC
```

It assumes you are not worried about where the index is positioned. You could use ABSTR to allocate on an absolute track value if you wanted to position the VTOC index, but remember that ABSTR will not work with SMS-managed volumes.

Although it is a relatively simple process, care must be taken when performing rebuilds of VTOC indexes because errors can result in inaccessible data.

John Bradley
Systems Programmer
Meerkat Computer Services (UK)

Catching up with COBOL

OVERVIEW

COBOL has been undergoing a silent evolution, and the IBM Enterprise COBOL for z/OS (Version 3.2) has brought new features that help integrate COBOL business processes and Web-oriented business processes by:

- Simplifying the componentization of COBOL programs and enabling interoperability with Java components.
- Promoting the exchange and usage of data in standardized formats, including XML and Unicode.
- Improved application development features in terms of functions, error handling and recursive calls.

IBM extensions range from a minor relaxation of rules to major capabilities, such as XML support, Unicode support,

object-oriented COBOL for Java interoperability, and DBCS character handling.

Note: IBM extensions generally add features, syntax, or rules defined by IBM and beyond those specified in ANSI and ISO COBOL standards.

In this article, the idea is to explore the application development feature improvements of COBOL. If you are a COBOL programmer with knowledge of C or PL/I, you will find these newer features of COBOL very familiar (we can call this 'COBOL catching up with other languages'!).

IMPROVED APPLICATION DEVELOPMENT

IBM, having improved inter-language communication with its Language Environment, is actually bringing the best of all worlds to COBOL. If you are familiar with PL/I, you will instantly recognize Intrinsic Functions and error handling using ON conditions. If you are not familiar, don't panic. We will cover these aspects in detail.

Intrinsic Functions

Similar to other high-level languages, Enterprise COBOL now has a set of functions – referred to as Intrinsic Functions – that provide capabilities for manipulating strings and numbers.

Examples

```
Unstring Function Upper-case(Name) Delimited By Space Into FirstName
LastName
Compute A = Function Log10(x)
Compute M = Function Max(a b c)
MOVE FUNCTION UPPER-CASE("change case") to dname.
```

A function-identifier is the combination of the COBOL reserved word FUNCTION followed by a function name (such as Max), followed by any arguments to be used in the evaluation of the function (such as x, y, z).

The following is a list of the functions supported:

- String handling – CHAR, DISPLAY-OF, LENGTH, LOWER-CASE REVERSE, UPPER-CASE.
- Mathematical formulae – ACOS, ASIN, ATAN, COS, FACTORIAL, INTEGER, INTEGER-OF-DATE, INTEGER-OF-DAY, INTEGER-PART, LOG, LOG10, MOD, RANDOM, REM, SIN, SQRT, SUM, TAN.
- Financial capabilities – ANNUITY, PRESENT-VALUE.
- Statistical functions – MAX, MEAN, MEDIAN, MIDRANGE, MIN, RANGE, STANDARD-DEVIATION, VARIATION.
- Date/time – CURRENT-DATE, DATE-OF-INTEGER, DATE-TO-YYYYMMDD, DATEVAL, DAY-OF-INTEGER, DAY-TO-YYYYMMDD, UNDATE, YEAR-TO-YYYY, YEARWINDOW.
- General – NATIONAL-OF, NUMVAL, NUMVAL-C, ORD, ORD-MAX, ORD-MIN, WHEN-COMPILED.

Functions are elementary data items, and return alphanumeric, national, numeric, or integer values. Functions cannot serve as receiving operands. The functions need not be defined in the DATA DIVISION because the value is derived automatically at the time of reference.

Use of arrays as arguments

When a function allows an argument to be repeated a variable number of times, you can refer to a table with the word ALL as a subscript. Specifying ALL as a subscript is equivalent to specifying all table elements possible using every valid subscript in that subscript position.

For example:

```
FUNCTION MAX(Table(ALL, ALL))
```

is equivalent to:

```
FUNCTION MAX(Table(1, 1) Table(1, 2) Table(1, 3)... Table(1, n)
Table(2, 1) Table(2, 2) Table(2, 3)... Table(2, n)
Table(3, 1) Table(3, 2) Table(3, 3)... Table(3, n)
```

```
.
.
.
Table(m, 1) Table(m, 2) Table(m, 3)... Table(m, n))
```

where *n* is the number of elements in the column dimension of *Table*, and *m* is the number of elements in the row dimension of *Table.*

The COBOL word function is a reserved word, but the function-names are not reserved. You can use them in other contexts, such as for the name of a variable. For example, you could use SQRT to invoke an intrinsic function and to name a variable in your program. Though probably not a good coding practice, this factor will be quite relevant in the context of existing programs.

Below are examples of using a function name as a variable:

- This allows Sqrt to be declared as a variable:

  ```
  Ø1 Sqrt Pic 99 value Ø.
  ```

- This refers to the SQRT variable declared:

  ```
  Compute Sqrt = 16 ** .5
  ```

- This refers to the intrinsic function SQRT:

  ```
  Compute z = x + Function Sqrt(y)
  ```

Nested functions

Functions that reference other functions as arguments are allowed as long as the results of the nested functions meet the requirements for the arguments of the outer function.

An example of a nested function is:

```
Compute x = Function Max((Function Sqrt(5)) 2.5 3.5)
```

Handling errors

COBOL contains special elements that help programmers to anticipate possible coding or system problems by putting code into the program to handle them. Such code is like built-

in distress flares or lifeboats – and the action can be handling the situation, issuing a message, or halting the program.

Handling errors in joining and splitting strings

During the joining or splitting of strings, the pointer used by STRING or UNSTRING might fall outside the range of the receiving field. A potential overflow condition exists, but COBOL does not let the overflow happen. Instead, the STRING or UNSTRING operation is not completed, the receiving field remains unchanged, and control passes to the next sequential statement.

If a STRING or UNSTRING statement does not have an ON OVERFLOW clause, the incomplete operation is not notified.

```
String Item-1 space Item-2 delimited by Item-3
into Item-4
with pointer String-ptr
on overflow
Display "A string overflow occurred"
End-String
```

In the above example, if *String-Ptr* has a value of zero, an overflow condition exists and the STRING operation will not be completed. If ON OVERFLOW had not been specified, you would not be notified that the contents of *Item-4* remain unchanged.

Handling errors in arithmetic operations

The results of arithmetic operations might be larger than the fixed-point field that is to hold them; another example of an error is division by zero. The ON SIZE ERROR clause after the ADD, SUBTRACT, MULTIPLY, DIVIDE, or COMPUTE statement can handle the situation.

An ON SIZE ERROR clause will be performed in the following cases and the result field will not be changed:

1 Fixed-point overflow – result larger than the fixed-point field.

2 Division by zero.

3 Zero raised to the zero power.

4 Zero raised to a negative number.

5 Negative number raised to a fractional power.

For example:

```
DIVIDE TOTAL-COST BY NUMBER-PURCHASED
GIVING ANSWER
ON SIZE ERROR
DISPLAY "ERROR IN DIVIDE-TOTAL-COST PARAGRAPH"
DISPLAY "SPENT " TOTAL-COST, " FOR " NUMBER-PURCHASED
PERFORM FINISH
END-DIVIDE
```

Handling errors in input and output operations

When an input or output operation fails, COBOL does not automatically take corrective action. Techniques for intercepting and handling certain input or output errors include End-of-file condition (AT END), file system return code, INVALID KEY phrase, imperative-statement phrases on your READ or WRITE statement, ERROR declaratives, and FILE STATUS clauses.

Code one or more ERROR declarative procedures in the declaratives section of your PROCEDURE DIVISION that will be given control if an input or output error occurs. The procedures can be:

1 A single common procedure for the entire program.

2 Procedures for each file open mode (whether INPUT, OUTPUT, I-O, or EXTEND).

3 Individual procedures for each particular file.

Here is an example of an ERROR declarative procedure (for a particular file):

```
PROCEDURE DIVISION.
DECLARATIVES
FILEA-ERROR SECTION
```

```
      USE AFTER STANDARD ERROR PROCEDURE ON FILEA
FILEA-ERROR.
      PERFORM FILEA-ERR-HANDLER.
```

Handling errors when calling programs

When a program dynamically calls a separately-compiled program, the called program might be unavailable to the system – the system could run out of storage or it could be unable to locate the load module. Normally the application will abend in such cases.

Use the ON EXCEPTION phrase on the CALL statement to perform your own error handling:

```
MOVE "REPORTA" TO REPORT-PROG
CALL REPORT-PROG
       ON EXCEPTION
              DISPLAY "Program REPORTA not available, using REPORTB.'
              MOVE "REPORTB" TO REPORT-PROG
              CALL REPORT-PROG
              END-CALL
END-CALL
```

User requested dumps

Dump the LE run-time environment and the member language libraries at any point in your program by calling the LE callable Service CEE3DMP (Call "CEE3DMP" Using Title-1, Options, Feedback-code).

For a system dump, request an abend without clean-up by calling the LE service CEE3ABD with a clean-up value of zero.

Writing routines for handling errors

Most error conditions that might occur can be handled by using the ON EXCEPTION phrase, the ON SIZE ERROR phrase, or other language constructs. But if an extraordinary condition like a machine check occurs, normally your application will abend.

However, Enterprise COBOL and Language Environment provide a way for a user-written program to gain control when

such conditions occur. Using LE condition handling, you can write your own error-handling programs in COBOL, which are loaded only when needed. They can report, analyse, or even fix up and allow your program to resume running.

Recursive CALLs

A called program can directly or indirectly execute its caller. For example, program X calls program Y, program Y calls program Z, and program Z then calls program X. This type of call is recursive.

To make a recursive call, you must code the RECURSIVE clause (IBM extension) on the PROGRAM-ID paragraph of the recursively called program.

If the optional RECURSIVE clause is specified, the program can be recursively re-entered while a previous invocation is still active:

1 The working-storage section of a recursive program defines storage that is statically allocated and initialized on the first entry to a program, and is available in a last-used state to any of the recursive invocations.

2 The local-storage section of a recursive program (as well as a non-recursive program) defines storage that is automatically allocated, initialized, and deallocated on a per-invocation basis. They are reinitialized to the value given in the VALUE clause. The local-storage section must begin with the header LOCAL-STORAGE SECTION.

3 The internal file connectors corresponding to FDs in the file section of a recursive program are statically allocated. The status of internal file connectors is part of the last-used state of a program that persists across invocations.

4 ALTER, GO TO without a specified procedure name, RERUN, USE FOR DEBUGGING, and SEGMENT-LIMIT are not supported in a recursive program.

CONCLUSION

Though COBOL has been undergoing lots of changes over the past few years, these features are not really being used by programmers, even for new development. The reason for this could be that COBOL programmers have been doing things in the old way for so long they are not even looking for these changes. The changes *per se* may be minor (in most cases, just an alternative way of doing things), but appropriate usage of them can make COBOL programming that much easier and also make your programs more elegant and improve their maintainability.

Sasirekha Cota
Tata Consultancy Services (India)

Job scheduling isn't just for job schedulers

INTRODUCTION

We have come a long way since the 1970s. Remember when job scheduling involved the preparation of run sheets or checklists that the operators would use to submit or release jobs at the appropriate time, usually when their predecessors had completed successfully?

As an operator in that era, I was weaned on massive card decks, the IBM 3505 card reader, and what seemed to be never-ending checklists of thousands of jobs and their dependencies. The Friday night checklist was the worst – double the number of jobs. The only saving grace was that we had most of the weekend to complete the work.

This system of job submission was slow and fraught with error. One job out of sequence could mean hours and hours of reruns. As the workload grew and the batch window shrank, automated systems became necessary.

Automated job scheduling systems have been common in the mainframe world since the early 1980s. They were developed to replace the manual and error-prone processes related to running the batch workload. Many of these scheduling solutions continued to evolve in the 1990s. Features and functions were added and some systems provided cross-platform capabilities. But are organizations using these feature-rich systems to their full potential?

Scheduling and running batch workload has always been regarded as a data centre-type of function. Even when automation came onboard, the prevailing attitude of data centre staff towards those outside the data centre was, “keep your hands off our scheduling system”.

While many organizations restrict the use of the job scheduling system to job schedulers and operations staff, others are exploring new ways to re-purpose their job scheduling system. This article explores some of these different ways. It shows you how to exploit the technology, reduce costs, and maximize your return on investment.

REPORTING

To effectively meet service level agreements, analysts need reporting facilities to measure performance and identify trends. For example, you may want to find out how many jobs have ABENDed with an S0C4 in the past week, how many payroll jobs were late in running last night, or which testing jobs are constantly consuming large amounts of CPU time.

Some scheduling products provide a set of canned reports and provide only internal information about schedule definitions. This is not sufficient for fine-tuning or for identifying problem areas. From an analytical point of view, you need access to much more information. You are not interested that Job B runs after Job A. Moreover, you are interested in actual SMF data. For example, you may be interested in the fact that Job B often runs late because Job A frequently ABENDs. Start

time, end time, CPU time, EXCP counts, completion codes, elapsed time, input-queue time are just a few of the fields on which you may want to report.

In the absence of adequate reporting facilities, analysts and developers tend to create their own reports, or they look to third-party products. Both of these approaches result in additional costs and can lead to maintenance problems. Who wants another system or product to maintain?

I've seen companies develop elaborate in-house reporting systems. These systems extract information from different sources (for example, the scheduling system, SMF datasets, an output management system, etc) and then feed data into other programs to provide statistics on batch runs. It is ironic to run hundreds of multi-step batch jobs in order to determine how to reduce the batch window.

With a powerful and versatile reporting facility included in your job scheduling system, you can eliminate the need for homegrown facilities and other tools. History reporting facilities can produce trend reports that identify recurring failures and aid in determining the root cause of errors.

The ability to report on jobs that the job scheduler did not submit is a real plus. The number of jobs submitted outside of the scheduling system can be significant, and this feature allows you to get the big picture.

Systems programmers, database administrators, operations analysts, and others can all benefit from reporting facilities. Simply set up your reports and use your scheduling system to schedule them.

REGRESSION TESTING OF PROGRAMS

Changes to programs are common. Whether it is a new system or a legacy system, programming changes need to be made. End users are always requesting enhancements and identifying problems that need to be corrected. As we know from experience, changes introduce problems.

When changes are made, programmers need to test and verify their new code, ensuring that no regression errors are introduced to the existing behaviour. Regression tests discover that new or changed code breaks what used to work. While that happens more often than any of us would like, most bugs are found in a program's new or intentionally changed behaviour.

Programmers may need to run regression job streams when making major changes. Without access to a scheduling system, this is a manual process. Often the developer manually submits one job, waits for it to complete, checks the output, and then manually submits the next job.

Why not allow the developers to set up and run their own job streams, or build them into the scheduling system and have the developers run them on-demand? This would reduce manual time-consuming processes and improve operational efficiencies.

TRANSLATION SERVICES

One of the main areas in which you may be able to employ your existing job scheduler is an area I refer to as translation services. Job scheduling information needs to be 'translated' from an application developer to a scheduler.

In many environments, an applications development team manually creates documents and graphs that represent job streams. This team normally consists of programmers or analysts who understand the jobs' relationships and other dependencies. These job flows are provided to another group, which is responsible for creating new applications and making changes to existing applications, for conversion into the job scheduling system. These flows may be members of a PDS, MS-Project flows, or rough sketches on pieces of paper.

For these types of change, a scheduling analyst must interpret the application owner's requests and make them understandable to the job scheduling system. Requests are

often poorly documented and require further explanation from the application developers.

With a graphical user interface to the job scheduling system, application owners can design their own job streams. Graphically, they outline the job flow and specify the JCL to execute, schedule frequencies, and other requirements. And they do this in terms that everybody will understand. The graphical user interface automatically creates the scheduling materials based on these requirements.

A straightforward, drag-and-drop methodology enables your application developers to generate production-ready, standard, application schedules. This allows you to:

- Reduce the time needed to take a business application from development to production.
- Improve productivity.
- Eliminate redundant work.
- Reduce errors and misunderstandings that arise during the hand-off from application developers to job schedulers to system operators.

TESTING SCHEDULES

Errors often occur during the nightly batch run and require immediate attention to avoid affecting service levels. These problems historically occur because test facilities are either inadequate or do not exist at all.

When applications are designed using a GUI, they generally will be tested first. If you don't have access to such a GUI, you need to ensure that job flows are set up and changed properly.

Do you have the ability to easily provide applications staff with their job flows in a simple format? Not having this presentation can result in problems that go unnoticed until it is too late. Printed copies of flowcharts and schedule definitions provide reviewers with easy-to-understand information, which they

can use to approve new and updated application definitions. They may even see that some streamlining can be done to improve batch processing.

An accurate workload simulation that quickly tests systems before they become part of the production workload is also needed. This allows you to test schedules before live runs and ensures that the correct jobs are being scheduled and in the correct sequence. You can even use the simulation results for change approvals.

The ability to see how new jobs interact with other jobs in a job flow facilitates bringing new processes into production. You can develop confidence that schedules will run accurately.

MIGRATING FROM TEST TO PRODUCTION

Moving new business applications into production is an error-prone process. The use of a graphical user interface to your scheduling system, as discussed earlier, reduces the risk of error. But there are other considerations and opportunities to efficiently and effectively migrate from test to production.

You need to focus on reusing work that you've already done. Let's say that a testing job flow has already been set up. What is so different about your production job flow? Usually the JCL library is different and the job names may need to be changed from a testing standard to a production standard. For example, a jobname may be UTOP5200 in the test environment but it becomes JTOP5200 in the production environment.

So you have some variable data. All the scheduling system needs to do is adjust the actual value of variable information based on the operating environment (for example test or production). You can then move job flows through your test phases and eventually to production essentially unchanged.

Some installations are able to use the same piece of JCL in both the testing and production environments. They simply use variables and other JCL tailoring techniques, provided by

the scheduling system, to ensure that the correct JCL is used in the appropriate environment.

Dynamic variable substitution leads to fewer opportunities to introduce errors when moving tested schedule definitions from development to test to production.

RUNNING AD HOC JOBS

Ad hoc job processing generally occurs when a user makes a call and a scheduling analyst initiates execution of the job. This may be a request to run a one-time job or it may be a request to run a job as part of a job flow. These jobs may be production jobs or testing jobs. Regardless, they are batch jobs that need to be scheduled on demand.

The architecture of your scheduling system needs to accommodate *ad hoc* scheduling requirements and provide a user-friendly interface. Without such a facility, customized interfaces are often developed and lead to more maintenance and support costs.

Different application teams may have leftover-type applications that include pre-defined JCL library names, alerts, condition code handling, etc. A user can simply insert the job to be run and specify the name of the job. For jobs with more complex requirements, a user may simply be able to issue a 'request' type of command, to indicate that the job should run when its requirements are met.

Some programmers may prefer to use TSO SUBMITs to submit their own batch jobs. While there is nothing wrong with this approach, there can be many advantages in using your scheduling system for such jobs. Even if they are stand-alone one-off jobs, some of the advantages provided by the scheduling system might include the following:

- Automatic addition of a rerun/restart step
- Additional reporting data

- Easier access to spool output
- Use of symbolic variables and other JCL-tailoring facilities
- Automatic notification for normal/abnormal processing
- Enhanced monitoring facilities.

There might be objections if every possible job needs to be defined to the scheduling system. Hopefully, this will not be the case. Users should be able to submit *ad hoc* work to which they have access without the need to have it pre-defined to the scheduling system.

You can reduce the need to interface with the scheduling group and improve productivity.

MONITORING WORKLOAD

IT people around the organization need to see what is scheduled, what is running, and when jobs are scheduled to finish. They may need to make phone calls or wait for e-mail replies to get the information they need – a waste of energy, time, and resources.

Things are often fuzzy at 4am, when you get called because a critical batch job has failed. You often need to see what impact that job has on the rest of the job flow.

Whether your scheduling system provides a graphical view or a text-based view of the batch workload, it is important to make this information available to all those with a vested interest. A common complaint among programmers, analysts, and other support staff is that they can't see what is going on.

Productivity can only be improved by allowing users access to the information that they need to do their jobs. They can monitor workload, obtain real-time status updates, review the critical path, handle exception conditions, and customize their own views to monitor the jobs in which they are interested.

You can increase efficiency by reducing phone calls and e-mails that are requesting status information.

OVERCOMING SECURITY CONCERNS

The biggest objection to opening up your job scheduling system to non-data centre staff is likely to be the fear of security exposures. Naturally, you cannot allow everybody access to everything. Security cannot be compromised. However, your fears can easily be overcome with the appropriate levels of security.

Tight integration between your job scheduling system and your existing mainframe-based security product is important. You want to be able to re-use existing definitions of users and groups. After all, providing a user with access to a job stream is really not much different from providing access to a dataset.

You will need to take advantage of a granular security interface so you can open up only what is necessary. You need to ensure that individual developers, schedulers, operators, and other users can perform only those tasks for which they are authorized, and that no individuals have access to anything to which they are not authorized.

In addition to the schedulers and operators that already have access, you will probably have many different types of user and security requirement. For example, some users will simply need to simulate and monitor certain business applications. Some users will need to control active jobs without changing permanent scheduling definitions. Others may need to define and run testing job streams, or to insert *ad hoc* jobs into the schedule.

CONCLUSIONS

We have come a long way since the manual job scheduling processes of the late 1970s. Since then, many data centres have invested in automated job scheduling systems to facilitate batch processing.

Automated job scheduling systems offer many benefits beyond data centre operations. Systems programmers, database administrators, operations analysts, applications

programmers, and end users are all potential users of a scheduling system. Whether you need to generate a historical report, test changes to the schedule, regression test new or changed programs, or translate business requirements into schedule definitions, your job scheduling system may be the perfect vehicle.

In the absence of adequate facilities to do their job, technical staff will develop their own systems or invest in even more products. This results in additional costs and maintenance problems.

While organizations are being asked to do more with less, you can actually do more with what you already have. You've made the investment. Why not realize the full potential of your existing solution and get the full value from your IT investment?

To maximize your return on investment, you should explore other ways of employing your existing system, or explore replacing your existing tool with one that can provide a better return on investment. Job scheduling isn't just for job schedulers any more.

Bob Pyette
Senior Product Specialist
Cybermation (USA)

Workload management

The MVS workload management provides a solution for managing workload distribution, workload balancing, and distributing resources to competing workloads. MVS workload management is the combined cooperation of various subsystems (CICS, JES, TSO/E, DDF, DB2, etc) with the MVS WLM component. With workload management, you define performance goals and assign a business importance to each

goal. You define the goals for work in business terms, and the system decides how much resource, such as CPU and storage, should be given to the work to meet its goal. WLM controls the dispatching priority based on the goals you supply. WLM raises or lowers the priority as needed to meet the specified goal. Thus, you do not need to fine-tune the exact priorities of every piece of work in the system and can focus instead on business objectives.

The WLM provides two modes of operation – the first one is compatibility mode, and the second one is the goal mode. In compatibility mode, threads are given a service class by the classification rules in the active WLM service policy. When you run in compatibility mode, you have to take on more performance management issues with stored procedures and user-defined functions that run in WLM-established address spaces. For example WLM cannot automatically start a new address space to handle additional high priority requests. In goal mode, threads are assigned a service class by the classification rules in the WLM service policy. Each service class period has a performance objective, and the workload manager raises or lowers that period's access to system resources as needed to meet the specified goal. When you run in goal mode, WLM automatically starts WLM-established address spaces for stored procedures and user-defined functions to help meet the service class goals you set.

My REXX procedure (TSO WLM) supports WLM administration. When the WLM definitions for an application environment are specified, you have to access the WLM through TSO panels. The procedure allows the following WLM commands:

- D WLM – display the current state of WLM (service policy name). You can see which policy is active.
- D WLM,APPLENV=* – display which application environments are defined.
- D WLM,APPLENV=DSNNWLM1 – display the status of a

particular application environment. An application environment initially enters the available state when the service policy that contains its definition is activated. ‘Available’ means the application environment is active for use, and servers are allowed to be started for it. Stored procedures and user-defined functions can be executed only in an available state.

- V WLM,APPLENV=DSNNWLM1,QUIESCE – the quiesce command causes workload management to request the termination of server address spaces for the application environment on completion of any active requests. You can issue a quiesce action for an application environment that is in the available state.
- V WLM,APPLENV=DSNNWLM1,RESUME – the resume command restarts an application environment that was previously quiesced and is in the quiesced state. It indicates to the workload manager that server address spaces can once again be started for this application environment. You also need to use the vary WLM command with the resume option when the application environment is in the unavailable state.
- V WLM,APPLENV=DSNNWLM1,REFRESH – the refresh command refreshes an application environment when you need to load a new version of a stored procedure or user-defined function. This command requests the termination of existing server address spaces and starts new ones in their place.
- F WLM,MODE=COMPAT – the modify command puts an application environment in compatibility mode.
- F WLM,MODE=GOAL – the modify command puts an application environment in goal mode.

WLM: REXX DRIVER PROCEDURE

```
/* REXX */
/* trace r */
```

```
zpfctl = 'OFF'
address ispexec 'vput (zpfctl) profile'
CUR='F1'
address ispexec "display panel(wlmpØ) cursor("CUR")"
do while rc=Ø
   if kurs='F1' | kurs='FIELD1' then do
        Call wlm1 'F1' field1
        CUR='F1'
   end
   if kurs='F2' | kurs='FIELD2' then do
        Call wlm1 'F2' field2
        CUR='F2'
   end
   if kurs='F3' | kurs='FIELD3' then do
        Call wlm1 'F3' field3
        CUR='F3'
   end
   if kurs='F4' | kurs='FIELD4' then do
        Call wlm1 'F4' field4
        CUR='F4'
   end
   address ispexec "display panel(wlmpØ) cursor("CUR")"
end
exit
```

WLM1: REXX PROCEDURE

```
/*rexx*/
/*trace r */
parse ARG poz text
zpfctl = 'OFF'
address ispexec 'vput (zpfctl) profile'
wait_time = 1Ø
Top:
"CONSOLE ACTIVATE"
lrc = rc
if lrc ¬= Ø
then do
   say "Message:"
   say "Unable to activate TSO CONSOLE"
   say "The return code:" lrc
   "CONSOLE DEACT"
   lrc = rc
   say "CONSOLE DEACT return code:" lrc
   "CONSOLE ACTIVATE"
   lrc = rc
   if lrc = Ø then say "Recovery successful!"
   else do
      say "Recovery attempt failed, return code:" lrc
```

```
      say "TSO CONSOLE authority ?"
      exit
   end
end
"CONSPROF SOLDISPLAY(NO) SOLNUM(1000)"
if poz = 'F1' then cmd="D WLM"
if poz = 'F2' then cmd="D WLM,APPLENV=*"
if poz = 'F3' then cmd="F WLM,MODE=COMPAT"
if poz = 'F4' then cmd="F WLM,MODE=GOAL"
address "TSO"
"CONSOLE SYSCMD("cmd")"
getcode = getmsg("msgs.","SOL",,,wait_time)
Call Get_code
address "TSO"
"CONSPROF SOLDISPLAY(YES) SOLNUM(1000)"
"CONSOLE DEACTIVATE"
if poz='F1' | poz='F3' | poz='F4'
then do
   address ispexec 'tbcreate "wlist" names(row)'
   do i = 1 to msgs.0
      row = strip(msgs.i)
      if length(row) > 80
      then do
         row=substr(strip(msgs.i),1,80)
         address ispexec 'tbadd "wlist"'
         row=substr(strip(msgs.i),81)
      end
      address ispexec 'tbadd "wlist"'
   end
   address ispexec 'tbtop "wlist"'
   title=text
   address ispexec 'tbdispl "wlist" panel(WLMP1)'
   if rc=8 then do
      address ispexec 'tbend "wlist"'
      Exit
   end
end
else do
   address ispexec 'tbcreate "vlist" names(detail)'
   do i = 3 to msgs.0
      detail = strip(msgs.i)
      address ispexec 'tbadd "vlist"'
   end
   address ispexec 'tbtop "vlist"'
   address ispexec 'tbdispl "vlist" panel(WLMP2)'
   if rc=8 then do
      address ispexec 'tbend "vlist"'
      Exit
   end
```

```
   if sel='R' | sel='r' then do
      title='Refresh Option'
      cmd='V WLM,APPLENV='||strip(word(detail,1))||',REFRESH'
      Call Action
   end
   if sel='A' | sel='a' then do
      title='Resume Option'
      cmd='V WLM,APPLENV='||strip(word(detail,1))||',RESUME'
      Call Action
   end
   if sel='Q' | sel='q' then do
      title='Quiesce Option'
      cmd='V WLM,APPLENV='||strip(word(detail,1))||',QUIESCE'
      Call Action
   end
   address ispexec 'tbend "vlist"'
   Signal Top
end
Action:
  wait_time = 10
  "CONSOLE ACTIVATE"
  "CONSPROF SOLDISPLAY(NO) SOLNUM(1000)"
  address "TSO"
  "CONSOLE SYSCMD("cmd")"
  getcode = getmsg("msgs.","SOL",,,wait_time)
  Call Get_code
  address "TSO"
  "CONSPROF SOLDISPLAY(YES) SOLNUM(1000)"
  "CONSOLE DEACTIVATE"
   address ispexec 'tbcreate "wlist" names(row)'
   do i = 1 to msgs.0
      row = strip(msgs.i)
      address ispexec 'tbadd "wlist"'
   end
   address ispexec 'tbtop "wlist"'
   address ispexec 'tbdispl "wlist" panel(WLMP1)'
   address ispexec 'tbend "wlist"'
   address ispexec 'tbend "vlist"'
   Signal Top
Return
Get_code:
  if getcode ¬=0
  then do
     say "GETMSG return code:" lrc
     "CONSPROF SOLDISPLAY(YES) SOLNUM(1000)"
     "CONSOLE DEACTIVATE"
     exit
  end
Return
Exit
```

WLMP0: MAIN PANEL

```
)attr default(%+_)
   [ type (output) intens(low)  color(green) caps(off)
   # type (output) intens(low)  color(white) caps(off)
   _ type (input)  intens(low)  color(yellow) caps(off) pad('_')
   + type (text)   intens(low)  color(green)
   / type (text)   intens(low)  color(yellow)
   ~ type (text)   intens(high) color(turquoise)
   @ type (text)   intens(high) color(red)   caps(off) hilite(reverse)
)body window(78,23) expand ($$)
/.....................................................................
                          + @ Work Load Manager +
/.....................................................................
%Command ===>_zcmd                                                    +
/.....................................................................
+
            + _z+[field1                                              +
            +
            + _z+[field2                                              +
            +
            + _z+[field3                                              +
            +
            + _z+[field4                                              +
+
/.....................................................................
+
                 #msg                                                 +
/PF3 - End +                                          ~Nov 2004,"ZB"
)init
  .ZVARS = '(f1 f2 f3 f4)'
  &field1 = 'Check the current state of WLM'
  &field2 = 'Display which application environments are defined'
  &field3 = 'Compatibility mode'
  &field4 = 'Goal mode'
  &msg = 'Place cursor on choice and press <Enter>'
  IF (&kurs = F1,FIELD1)
      .attr (field1) = 'color (yellow) caps(on)'
  IF (&kurs = F2,FIELD2)
      .attr (field2) = 'color (yellow) caps(on)'
  IF (&kurs = F3,FIELD3)
      .attr (field3) = 'color (yellow) caps(on)'
  IF (&kurs = F4,FIELD4)
      .attr (field4) = 'color (yellow) caps(on)'
)proc
  &kurs = .CURSOR
  if (.pfkey = pf03) &pf3 = exit
)end
```

WLMP1: MESSAGE PANEL

```
)Attr Default(%+_)
   $ type(output) intens(high) caps(off) color(yellow)
   # type(text)   intens(high) caps(off) hilite(reverse)
   { type(output) intens(low ) caps(off) just(asis ) color(blue)
   ^ type(output) intens(low ) caps(off) just(asis ) color(green)
)Body  Expand(//)
%-/-/- $title                                                    +%-/-/-
%Command ===>_zcmd                                  / /%Scroll ===>_amt +
%WLM Command:{cmd
+----------------------------- Message ------------------------------
)Model
^z
)Init
  .ZVARS = '(row)'
  &amt = PAGE
)Reinit
)Proc
)End
```

WLMP2: SELECTION RESULT PANEL

```
)Attr Default(%+_)
   ! type(text)   intens(high) caps(on ) color(yellow)
   ? type(text)   intens(high) caps(on ) color(green) hilite(reverse)
   # type(text)   intens(high) caps(off) hilite(reverse)
   ] type(input)  intens(high) caps(on ) just(left ) pad('-')
   ^ type(output) intens(low ) caps(off) just(asis ) color(green)
)Body  Expand(//)
%-/-/- ? Selection Result +%-/-/-
%Command ===>_zcmd                                  / /%Scroll ===>_amt +
+-------------------------------------------------------------------------
+Valid cmd:!R+Refresh !A+Activate-Resume !Q+Quiesce
+Enter Valid cmd and press!Enter+
!PF3+Return
+-------------------------------------------------------------------------
#cmd#Application Environment Name    #State     State Data
)Model
+]z+^z
)Init
  .ZVARS = '(sel detail)'
  &amt = PAGE
  &sel = ''
)Reinit
)Proc
)End
```

Bernard Zver (bernard.zver@informatika.si)
DBA
Informatica (Slovenia)

ASG has announced ASG-SmartTune, its performance monitoring and tuning solution for mainframe applications.

ASG-SmartTune measures and analyses the performance characteristics of mainframe applications, subsystems, and jobs in test and production environments. It provides both real-time and historical performance statistics and the ability to drill down to and view actual lines of code, enabling developers and analysts to pinpoint areas within an application that require tuning.

For further information contact:
URL: www.asg.com/newsroom/pr_details.asp?id=151.

***

Compuware has announced Version 3.1 of File-AID/CS, its data testing tool.

Using the new version, development teams can populate fields with millions of unique names, addresses, social security numbers, phone numbers, and many other values by using simple drag-and-drop functionality. Additionally, data can be generated from a pre-packaged data library of unique values or customized to reflect an organization's unique naming standards or conventions.

File-AID/CS supports Oracle, Microsoft SQL Server, DB2 UDB, Sybase, XML, VSAM, IMS, and DB2 UDB for z/OS.

For further information contact:
URL: www.compuware.com/products/fileaid/cs.htm.

***

Vanguard Integrity Professionals has announced Version 5.3 of Vanguard Security Solutions.

The product provides customers with enterprise-wide security on demand in cross-platform environments. This newest release includes a new product, Vanguard ez/Token, a two-factor authentication solution that integrates token technology with RACF. Vanguard Security Solutions 5.3 offers more than 40 reporting, auditing, and efficiency enhancements.

For further information contact:
URL: www.go2vanguard.com/docs/marketing/press_releases/FINAL%20Press%20Release%201.6.pdf.

***

TeamQuest has announced Version 9.2 of TeamQuest Performance Software for IBM eServer pSeries, zSeries, and iSeries server platforms. In addition to z/OS, OS/400, and i5/OS support, enhancements in TeamQuest Performance Software 9.2 include agents for VMware ESX server and network device, network applications and WebLogic. The network application agents provide end-to-end performance monitoring without installing client agents.

TeamQuest Performance Software monitors all the components in a data centre, identifying deviation from normal and locating the components of interest.

For further information contact:
URL: www.teamquest.com/pressroom/pr/0105.shtml.