202

MVS update

July 2003

In this issue

MVS Update

Published by
Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38342
From USA: 01144 1635 38342
E-mail: trevore@xephon.com

North American office
Xephon
PO Box 350100
Westminster, CO 80035-0100
USA
Telephone: 303 410 9344

Subscriptions and back-issues
A year's subscription to *MVS Update*, comprising twelve monthly issues, costs £340.00 in the UK; $505.00 in the USA and Canada; £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 1999 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.

Editor
Trevor Eddolls

Contributions
When Xephon is given copyright, articles published in *MVS Update* are paid for at the rate of £170 ($260) per 1000 words and £100 ($160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 ($80) per 100 lines. In addition, there is a flat fee of £30 ($50) per article. 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.

Execute program with extended parameter

PROBLEM ADDRESSED

The EXEC parameter has always been a useful and simple means of passing parameters to a program; rather than processing a file, five instructions suffice to access the EXEC parameter. In particular for compilers, there has been a dramatic increase in the number and range of parameters. However, since time immemorial (with regard to z/OS and its predecessors), the maximum length of the EXEC parameter that can be specified by JCL has remained at 100 characters. This may have been appropriate when (real) memory was literally worth its weight in gold, but nowadays this restriction is purely artificial. No robust program should have any problem in handling an EXEC parameter of any specified length (the infamous buffer overflow problem with the associated security risks known from the Windows world underscores the problems associated with non-robust programs). Indeed, even some programs which explicitly state that they can process only parameter lists with a maximum of 100 characters, when invoked dynamically can actually process longer EXEC parameter lists (eg some COBOL compilers). A second problem concerns the somewhat abstruse rules for the continuation of a JCL EXEC PARM when commas, parentheses, and apostrophes are involved.

Some, but not all, compilers solve these problems by allowing the use of an options file that can contain additional compiler options.

EXTDPARM, the program described in this article, allows an extended EXEC parameter, specified in the SYSPARM file, to be passed to a program invoked dynamically. The name of the program to receive control is specified in the EXTDPARM EXEC parameter. This program is loaded from the current load library (STEPLIB, LOADLIB, etc). Thus, EXTDPARM can be used to invoke any program with an extended parameter provided the invoked program correctly processes its EXEC parameter.

SOLUTION

EXTDPARM dynamically invokes the program specified in the EXEC parameter with the parameter formed from the input specified by the SYSPARM file. The trailing blanks in each line of the SYSPARM file are removed. The resulting parameter list is formed by concatenating each line of the SYSPARM file; any special characters and leading blanks are passed unchanged.

Example 1:

```
ALPHA BETA
  GAMMA
    DELTA
```

produces the parameter list:

```
ALPHA BETA  GAMMA    DELTA
```

Example 2:

```
ALPHA BETA,
  'GAMMA '
    DELTA
```

produces the parameter list:

```
ALPHA BETA,  'GAMMA '    DELTA
```

Invocation:

```
//        EXEC PGM=EXTDPATM,PARM=pgmname
//STEPLIB DD DSN=loadlib,DISP=SHR
//SYSPARM DD *
parm1
parm2
...
```

DD statements are:

- pgmname – the name of the program to be invoked.
- loadlib – the name of the load library that contains EXTDPATM and pgmname. If required, additional load libraries can be concatenated.
- parm1 ... parmn – the parameters to be passed to pgmname.

Note: although SYSPARM is shown here assigned to the input stream, it can be assigned to any PS dataset that has RECFM=FB and LRECL<256 as file attributes.

Unless EXTDPATM detects a processing error – eg no program name specified (or name too long), or specified parameter too long (> 32760 characters, program constant) – it sets its completion code to that returned from the invoked program.

Error returns:

- -1 – no external program name specified or name longer than eight characters.
- -2 – parameter overflow.

EXTDPATM

```
         TITLE 'Execute Program With Extended Parameter'
**
* EXTDPATM: Execute (load) program with extended parameter    **
Invocation:
*  //        EXEC EXTDPATM,PARM=execname
*  //SYSPARM DD * (RECFM=FB,LRECL<256)
*
*  DD:SYSPARM contains the parameters to be passed to the
*  specified program. Trailing blanks are removed from each
*  line. The processed lines are concatenated together (any
*  required delimiters must be specified explicitly, eg
*  commas, leading blanks).
*
* Return:
*   Return code from the executed program
* Error returns:
*   -1: no external program name specified or name longer than
*       8 characters.
*   -2: parameter overflow
**
         PRINT NOGEN
         SPACE 1
EXTDPARM CSECT
EXTDPARM AMODE 31
EXTDPARM RMODE 24
* initialise addressing
         STM   R14,R12,12(R13)        save registers
         BASR  R12,Ø                  base register
         USING *,R12
         LA    R15,SA                 A(save-area)
         ST    R13,4(R15)             backward ptr
         ST    R15,8(R13)             forward ptr
         LR    R13,R15                A(new save-area)
         SPACE 1
         LHI   R15,-1                 preload ReturnCode register
         L     R2,Ø(R1)               pointer to parameters
         SR    R1,R1                  zeroise R1
         ICM   R1,3,Ø(R2)             length of program name
         JZ    EXIT                   no exec name (=error)
         CHI   R1,L'EXECNAME          test length
         JH    EXIT                   too long (=error)
         BCTR  R1,Ø                   LengthCode(parm)
         EX    R1,EXMOVE              store program name
         SPACE 1
         OPEN  (SYSPARM,(INPUT))      open SYSPARM file
         LTR   R15,R15                test OPEN return code
         JNZ   NOPARM                 open error -> file does not exist
         LA    R2,EXECDATA
         LR    R3,R2
         AHI   R3,DATALEN             end of exec data
         USING IHADCB,SYSPARM
READLOOP LHI   R15,-2                 preload ReturnCode
         CR    R2,R3                  test for buffer overflow
         JNL   EXIT                   buffer overflow
         GET   SYSPARM,(R2)           read logical record
* remove trailing blanks
         LH    R1,DCBLRECL            logical record length
         AR    R2,R1                  address of record-end +1
TESTLOOP BCTR  R2,Ø                   decrement address pointer
         CLI   Ø(R2),C' '             test for blank
         JNE   LOOPOUT                non-blank found
         JCT   R1,TESTLOOP            test next character
* empty record
LOOPOUT  LA    R2,1(R2)               correct address pointer
         J     READLOOP               read next record
         SPACE 1
FILEEOF  DS    ØH                     EOF(DD: SYSPARM)
         S     R2,=A(EXECDATA)        length of buffer data
         STH   R2,EXECLEN             save length
         CLOSE (SYSPARM)
         SPACE 1
NOPARM   DS    ØH                     load and invoke program
         LINK  EPLOC=EXECNAME,PARAM=(CALLPARM),VL=1
         SPACE 1
EXIT     DS    ØH                     job end
         L     R13,4(R13)             restore addr. of old save-area
         RETURN (14,12),RC=(15)
         SPACE 1
EXMOVE   MVC   EXECNAME(Ø),2(R2)
         SPACE 1
```

```
* symbolic register equates
RØ       EQU   Ø
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R1Ø      EQU   1Ø
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         TITLE 'Data Areas'
         LTORG
         SPACE 1
SYSPARM  DCB   DDNAME=SYSPARM,DSORG=PS,MACRF=GM,EODAD=FILEEOF
         SPACE 1
SA       DS    18F                      register save area
         SPACE 1
EXECNAME DC    CL8' '                   program name
         SPACE
DATALEN  EQU   3276Ø
CALLPARM DS    ØH
EXECLEN  DC    H'Ø'
EXECDATA DS    CL(DATALEN)
         DS    CL256                    padding
         SPACE 1
         DCBD  DSORG=PS,DEVD=DA
         END
```

SAMPLE PROGRAM

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COBPARM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
Ø1 EXECPARM.
 Ø2 EXECPARM-LEN PIC 9(4) BINARY.
 Ø2 EXECPARM-DATA PIC X(3276Ø).
PROCEDURE DIVISION USING EXECPARM.
    DISPLAY 'EXECPARM:' EXECPARM-DATA
    DISPLAY 'EXECPARM-LEN:' EXECPARM-LEN
    STOP RUN.
```

SAMPLE INVOCATION

```
//        EXEC PGM=EXTDPATM,PARM=TESTPGM
//STEPLIB DD DSN=loadlib,DISP=SHR
//SYSPARM DD *
NOADATA  ADV
 NOANALYZE  APOST
  ARITH(COMPAT) NOAWO
  BUFSIZE(8192)
```

OUTPUT

```
EXECPARM:NOADATA  ADV NOANALYZE  APOST  ARITH(COMPAT) NOAWO
BUFSIZE(8192)
EXECPARM-LEN:ØØ65
```

Systems Programmer
(Germany) © Xephon 2003

Subroutine to get date and time

TIMEDATE is a program intended to be used as a subroutine for other programs that need the current date and time in character format. The calling program should pass TIMEDATE an area with 14 bytes. On return, that area contains the local date and time in the format YYYYMMDDhhmmss.

```
*===================================================================
*
* TIMEDATE - Returns current date and time in format YYYYMMDDhhmmss.
*            The calling program should pass a 14 byte area.
*
*===================================================================
&PROG    SETC  'TIMEDATE'
&PROG    CSECT
&PROG    AMODE 31
&PROG    RMODE 24
         STM   R14,R12,12(R13)
         LR    12,15
         USING &PROG,R12
         ST    R13,SAVEA+4
         LA    R11,SAVEA
         ST    R11,8(R13)
         LR    R13,R11
         LR    R2,R1
         L     R3,Ø(Ø,R2)
         B     CONTINUE
         DC    CL16' TIMEDATE V.2.Ø '
         DC    CL8'&SYSDATE'
*
CONTINUE EQU   *
         TIME  DEC,HOR1,                                               X
               ZONE=LT,                                                X
               LINKAGE=SYSTEM,                                         X
               DATETYPE=YYYYMMDD
*
         UNPK  UNPH9,HOR1
         NC    UNPH8,XTR1
         TR    UNPH8,XTR2
*
         UNPK  UNPD9,DAT1
         NC    UNPD8,XTR1
         TR    UNPD8,XTR2
*
         MVC   UNPD8+8(6),UNPH8+2
         MVC   Ø(14,R3),UNPD8
*
EXIT     EQU   *
         L     R13,SAVEA+4
         LM    R14,R12,12(13)
         SR    R15,R15
         BR    R14
*
SAVEA    DS    18F
*
HOR1     DS    CL4
         DS    CL4
DAT1     DS    CL5
         DS    CL3
*
         DS    ØD
UNPH9    DS    ØCL9
UNPH8    DS    CL8
         DS    C
*
         DS    ØD
UNPD9    DS    ØCL9
UNPD8    DS    CL8
         DS    CL1Ø
*
         DS    ØF
XTR1     DC    X'ØFØFØFØFØFØFØFØF'
```

```
XTR2     DC    C'Ø123456789ABCDEF'
*
         YREGS
         END
```

Systems Programmer
(Portugal) © Xephon 2003

System layout verification tool

For quite some time now, a typical mainframe installation environment comprises multiple CPUs with many partitions active.

The frequency with which the core software needs to be maintained imposes a precise operating system design that needs to clearly reflect the division between 'target', 'distribution', and 'operational' dataset types.

Generally speaking, a 'target' or 'distribution' dataset is one that has a 'DDDEF' definition in its respective SMP/E zone, whereas 'operational' datasets are all those that contain the customizations of a particular product or feature.

Moreover, the contents of the 'target' and 'distribution' datasets may change completely with every upgrade of the operating system, whereas the 'operational' datasets would have minimal updates if any, and so are maintained intact with each and every release of the core software.

This differentiation of datasets isn't merely academic, but rather becomes almost obligatory if one intends to have an operating system and base software that must be easy to upgrade and maintain.

The fundamental characteristics of this software design are effectively summed up in the following technical documents:

- *OS/390 Maintenance Philosophy – An IBM View.*

- *SHARE SUMMER 2000 Technical Conference Session 2825, 26 July 2000.*

Even the Serverpac dialogs push the systems programmer to dedicate the RESVOL volumes only to the ‘target’ datasets, while the SMP/E and ‘distribution’ datasets have separate volumes.

It is desirable that those ‘operational’ datasets that define the ‘image customization set’ get allocated on a different volume from the ones that go to a new ServerPac or CBPDO – in this case, one is often advised to take advantage of indirect cataloguing through the use of the system symbols that are defined in the member of SYS1.PARMLIB(IEASYM*xx*).

Each one of us is full of good intentions, but in the end even in a job well done the odd error can slip through and in time create problems.

To this end I have written EXAMSYS, a batch utility that will examine (one volume at a time) all those volumes intended for the operating system.

The end result for each selected volume is a printout listing every file, clearly pointing out the relating SMP/E zone, the type of cataloguing used as well as the correct type to be used, and also the quantity of ‘operational’ datasets on that volume.

Having read through the report, one ultimately understands whether or not that particular volume has been correctly assigned and if any datasets have been allocated on the wrong logical volume.

To do all this I have used the standard IBM utilities as well as a few lines of code using the REXX language.

Everything can be easily modified, should one so desire, including the structure of the final printout (see Step/JCL with ICETOOL).

EXAMDSN REXX SAMPLE

```
/* REXX ---------------------------------- */
/* REXX RECORD MANIPULATION OF PRINTOUT FILE */
/* REXX ------------------------------------ */
  TRACE OFF
  " PROF NOPREF "
   WK_FILEV  = 'START'
   WK_FILED  = 'START'
   WK_RCD = Ø
   WK_RCV = Ø
  "EXECIO Ø DISKR FILED (OPEN"
  "EXECIO Ø DISKR FILEV (OPEN"
  "EXECIO Ø DISKW FILEP (OPEN"
/* - EMPTY FILE ------------------------- */
  IF WK_FILED = 'START' THEN DO
      "EXECIO 1 DISKR FILED "
       WK_RCD = RC
           IF WK_RCD > Ø THEN EXIT(98)
       PULL WK_FILED
                            END
  IF WK_FILEV = 'START' THEN DO
      "EXECIO 1 DISKR FILEV "
       WK_RCV = RC
           IF WK_RCV > Ø THEN EXIT(99)
       PULL WK_FILEV
                            END
/* - EMPTY FILE ------------------------- */

     DO FOREVER

       SELECT

    WHEN WK_RCD > Ø
       THEN    DO
       CALL PROC_PRINT
      "EXECIO 1 DISKR FILEV "
       WK_RCV = RC
           IF WK_RCV > Ø THEN LEAVE
       PULL WK_FILEV
               END

    WHEN SUBSTR(WK_FILEV,Ø2,44) = SUBSTR(WK_FILED,3Ø,44)
       THEN    DO
        CALL PROC_PRINT
       "EXECIO 1 DISKR FILED "
        WK_RCD = RC
        PULL WK_FILED
       "EXECIO 1 DISKR FILEV "
        WK_RCV = RC
           IF WK_RCV > Ø THEN LEAVE
        PULL WK_FILEV
               END
    WHEN SUBSTR(WK_FILEV,Ø2,44) > SUBSTR(WK_FILED,3Ø,44)
       THEN    DO
      "EXECIO 1 DISKR FILED "
       WK_RCD = RC
       PULL WK_FILED
               END

    WHEN SUBSTR(WK_FILEV,Ø2,44) < SUBSTR(WK_FILED,3Ø,44)
       THEN    DO
       CALL PROC_PRINT
      "EXECIO 1 DISKR FILEV "
       WK_RCV = RC
           IF WK_RCV > Ø THEN LEAVE
       PULL WK_FILEV
               END

       OTHERWISE NOP
       END
     END

/* ------------------------------------- */
   "EXECIO Ø DISKR FILEV (FINIS"
   "EXECIO Ø DISKR FILED (FINIS"
   "EXECIO Ø DISKW FILEP (FINIS"

    EXIT(ØØ)

/* INTERNAL ROUTINE --------------------- */
  PROC_PRINT:

    SELECT
    WHEN INDEX(WK_FILEV,'.VTOCIX.')   <> Ø THEN RETURN
    WHEN INDEX(WK_FILEV,'.VVDS.')     <> Ø THEN RETURN
    WHEN INDEX(WK_FILEV,'.DATA ')     <> Ø THEN RETURN
    WHEN INDEX(WK_FILEV,'.INDEX ')    <> Ø THEN RETURN
    WHEN INDEX(WK_FILEV,'.CATINDEX ') <> Ø THEN RETURN
    OTHERWISE NOP
    END

/* ------------------------------------- */
    WK_DSNP =  SUBSTR(WK_FILEV,Ø2,44)
    WK_ZON  =  'OPERATIONAL    '
    WK_DRC  =  ' NO  '
    WK_CAT  =  ' NO  '
    WK_FLG  =  '<<'
    WK_DDN  =  '        '

/* ------------------------------------- */
             LL = OUTTRAP(LINE.)
    "LISTC ENT("WK_DSNP") VOL"
    IF RC = Ø THEN DO
         WK_CAT = ' YES '
         I = 4
         DO UNTIL I = 9
           IF INDEX(LINE.I,"DEVTYPE------X'ØØØØØØØØ'") > Ø
           THEN WK_DRC = ' YES '
         I = I + 1
         END
                   END

    LL = OUTTRAP(OFF)

/* ----------------------------------- */
    SELECT
      WHEN  WK_RCD > Ø THEN NOP
      WHEN  SUBSTR(WK_FILEV,Ø2,44) = SUBSTR(WK_FILED,3Ø,44)
         THEN DO
              WK_ZON =  SUBSTR(WK_FILED,1Ø1,15)
              WK_DDN =  SUBSTR(WK_FILED,Ø2,Ø8)
              END
      OTHERWISE NOP
    END

/* ----------------------------------- */
    WK_FILEP = WK_DSNP WK_DDN WK_ZON WK_DRC WK_CAT WK_FLG
    PUSH WK_FILEP
   "EXECIO 1 DISKW FILEP "

/* ----------------------------------- */
    RETURN
```

EXAMOUT REXX SAMPLE

```
/* REXX ------------------------------- */
/* REXX RECORD MANIPULATION OF OUTPUT FILE */
/* REXX ------------------------------- */
  TRACE OFF
   WK_EMPTY = 'NOK'
   WK_ZONE  = ''
  "EXECIO Ø DISKR INPØØØ (OPEN"
  "EXECIO Ø DISKW SORTIN (OPEN"
/* ----------------------------------- */
  READØØ:

  DO FOREVER
   "EXECIO 1 DISKR INPØØØ "
    WK_RC = RC
    IF WK_RC > Ø THEN LEAVE
    PULL MY_INPØØØ

     IF INDEX(MY_INPØØØ,'DDDEF ENTRIES') = Ø
          THEN    DO
            WK_SORTIN = SUBSTR(MY_INPØØØ,Ø1,1ØØ)WK_ZONE
            PUSH WK_SORTIN
           "EXECIO 1 DISKW SORTIN "
                  END
          ELSE    DO
            WK_EMPTY = 'OK'
            WK_ZONE  =  SUBSTR(MY_INPØØØ,Ø2,Ø7)
                  END
END

ENDØØ:

"EXECIO Ø DISKR INPØØØ (FINIS"
"EXECIO Ø DISKW SORTIN (FINIS"
/* -------------------------------------- */
 IF WK_EMPTY = 'NOK' THEN EXIT(2Ø)
/* -------------------------------------- */
 IF WK_RC  <> 2      THEN EXIT(5Ø)
/* -------------------------------------- */
 ADDRESS  "LINKMVS" "ICEMAN"
 IF RC     <> Ø      THEN EXIT(4Ø)
/* -------------------------------------- */
"EXECIO Ø DISKW WORKEND (OPEN"
"EXECIO Ø DISKR SORTOUT (OPEN"
 WK_REC1 = '$$.START.$$'
 WK_REC2 = ''

 DO FOREVER
    SELECT
 WHEN WK_REC1 = '$$.END.$$'
     THEN  LEAVE

 WHEN WK_REC1 = '$$.START.$$'
     THEN CALL READ_ALL

 WHEN SUBSTR(WK_REC1,2,44) = SUBSTR(WK_REC2,2,44)
     THEN DO
     WK_ZON1   = SUBSTR(WK_REC1,1Ø1,7)
     WK_ZON2   = SUBSTR(WK_REC2,1Ø1,7)
     WK_WORKEND = SUBSTR(WK_REC1,Ø1,Ø99) WK_ZON1 WK_ZON2
     PUSH WK_WORKEND
     "EXECIO 1 DISKW WORKEND "
     CALL READ_ALL
          END

 WHEN SUBSTR(WK_REC1,2,44) <> SUBSTR(WK_REC2,2,44)
```

```
       THEN DO
       WK_WORKEND = WK_REC1
       PUSH WK_WORKEND
       "EXECIO 1 DISKW WORKEND "
       WK_REC1 = WK_REC2
       CALL READ_ONE
            END
   OTHERWISE NOP
      END
   END

  "EXECIO Ø DISKR SORTOUT (FINIS"
  "EXECIO Ø DISKW WORKEND (FINIS"
   EXIT(ØØ)

/* INTERNAL ROUTINE --------------------- */
  READ_ALL:
  "EXECIO 1 DISKR SORTOUT"
     IF RC > Ø THEN WK_REC1 = '$$.END.$$'
        ELSE  PULL WK_REC1

  READ_ONE:
  IF WK_REC1 = '$$.END.$$' THEN RETURN

  "EXECIO 1 DISKR SORTOUT"
     IF RC > Ø THEN WK_REC2 = '$$.END.$$'
        ELSE  PULL WK_REC2

   RETURN
```

SAMPLE JCL TO RUN EXAMSYS

```
//......  JOB ......,......,CLASS=.,MSGCLASS=.,REGION=ØM,COND=(Ø,GT)
//** ---------------------------------------------------------- **
//** BEFORE SUBMITTING THE JOB:                                  **
//** > IN ST1ØØ INDICATE:                                        **
//**   . THE CORRECT CSI  NAME  ON THE 'SMPCSI'  DD CARD;        **
//**   . THE CORRECT ZONE NAMES ON THE 'SMPCNTL' DD CARD.        **
//** > IN ST4ØØ INDICATE:                                        **
//**   . THE NAME OF THE VOLUME TO BE EXAMINED IN 'SYSIN' CARD.**
//**   . THE CORRECT VOL=SER ON THE 'DISKØØ' DD CARD.            **
//** ---------------------------------------------------------- **
//ST1ØØ    EXEC PGM=GIMSMP
//SMPCSI   DD   DISP=SHR,DSN='your_.GLOBAL.CSI'
//SMPLIST  DD   DSN=&&SMPLIST,DISP=(MOD,PASS),UNIT=VIO,
//  SPACE=(CYL,3),DCB=(LRECL=121,BLKSIZE=726Ø,RECFM=FBA)
//SYSOUT   DD   SYSOUT=*
//SMPLOG   DD   DUMMY
//SMPLOGA  DD   DUMMY
```

```
//SMPRPT   DD   DUMMY
//SMPOUT   DD   DUMMY
//SMPCNTL  DD   *
  SET BOUNDARY(GLOBAL).
      LIST DDDEF.
  SET BOUNDARY(your_target_zone).
      LIST DDDEF.
  SET BOUNDARY(your_distribution_zone).
      LIST DDDEF.
/*
//** --------------------------------------------------------- **
//ST2ØØ    EXEC PGM=ICEMAN
//SORTIN   DD   DSN=&&SMPLIST,DISP=(OLD,PASS)
//SORTOUT  DD   DSN=&&DDDEF1,DISP=(,PASS),UNIT=VIO,
//  SPACE=(CYL,3),DCB=*.ST1ØØ.SMPLIST
//SYSOUT   DD   DUMMY
//SYSIN    DD   *
 SORT FIELDS=COPY
 INCLUDE COND=(12,17,CH,EQ,C'DATASET         =',                      *
            OR,1Ø,13,CH,EQ,C'DDDEF ENTRIES')
/*
//** --EXEC REXX - EXAMOUT ----------------------------------- **
//ST3ØØ    EXEC PGM=IKJEFTØ1,PARM='%EXAMOUT'
//INPØØØ   DD   DSN=&&DDDEF1,DISP=(OLD,PASS)
//SORTIN   DD   DSN=&&DDDEF2,DISP=(,PASS),UNIT=VIO,
//  SPACE=(CYL,3),DCB=*.ST1ØØ.SMPLIST
//SORTOUT  DD   DSN=&&DDDEF3,DISP=(,PASS),UNIT=VIO,
//  SPACE=(CYL,3),DCB=*.ST1ØØ.SMPLIST
//WORKEND  DD   DSN=&&DDDEND,DISP=(,PASS),UNIT=VIO,
//  SPACE=(CYL,3),DCB=*.ST1ØØ.SMPLIST
//SYSPROC  DD   DISP=SHR,DSN=your_sysproc
//SYSPRINT DD   SYSOUT=*
//SYSTSPRT DD   SYSOUT=*
//SYSOUT   DD   DUMMY
//SYSTSIN  DD   DUMMY
//SYSIN  DD *
  SORT FIELDS=(3Ø,44,CH,A)
/*
//** --------------------------------------------------------- **
//ST4ØØ    EXEC PGM=IEHLIST
//SYSPRINT DD   DSN=&&VTOCL,DISP=(,PASS),UNIT=VIO,
//   SPACE=(CYL,3),DCB=*.ST1ØØ.SMPLIST
//DISKØØ   DD   UNIT=339Ø,VOL=SER=your_volser,DISP=SHR
//SYSIN    DD *
 LISTVTOC  VOL=339Ø=your_volser
/*
//** --------------------------------------------------------- **
//ST5ØØ    EXEC PGM=ICEMAN
//SORTIN   DD   DSN=&&VTOCL,DISP=(OLD,PASS)
//SORTOUT  DD   DSN=&&VTOCP,DISP=(,PASS),UNIT=VIO,
//  SPACE=(CYL,3),DCB=*.ST1ØØ.SMPLIST
//SYSOUT   DD   DUMMY
//SYSIN    DD   *
 SORT FIELDS=COPY
 OMIT    COND=(Ø2,Ø6,CH,EQ,C'DATE: ',                             *
            OR,Ø3,1Ø,CH,EQ,C'THERE ARE ',                          *
            OR,Ø5,12,CH,EQ,C'THERE IS A  ',                        *
            OR,Ø5,13,CH,EQ,C'DATA SETS ARE',                       *
            OR,15,17,CH,EQ,C'--DATA SET NAME--',                   *
            OR,18,24,CH,EQ,C'CONTENTS OF VTOC ON VOL ',            *
            OR,32,25,CH,EQ,C'SYSTEMS SUPPORT UTILITIES')
/*
//** --EXEC REXX - EXAMDSN ------------------------------- **
//ST6ØØ    EXEC PGM=IKJEFTØ1,PARM='%EXAMDSN'
//FILEV    DD   DSN=&&VTOCP,DISP=(OLD,PASS)
//FILED    DD   DSN=&&DDDEND,DISP=(OLD,PASS)
//FILEP    DD   DSN=&&OUTPØ,DISP=(,PASS),UNIT=VIO,
//  SPACE=(CYL,3),DCB=(LRECL=9Ø,BLKSIZE=6Ø3Ø,RECFM=FB)
//SYSPRINT DD   DUMMY
//SYSTSPRT DD   DUMMY
//SYSPROC  DD   DISP=SHR,DSN=*.ST3ØØ.SYSPROC
//SYSTSIN  DD   DUMMY
//** ------------------------------------------------------ **
//ST7ØØ    EXEC PGM=ICETOOL
//TOOLMSG  DD   DUMMY
//DFSMSG   DD   DUMMY
//IN1      DD   DISP=(OLD,PASS),DSN=&&OUTPØ
//REPORT   DD   SYSOUT=*
//TOOLIN   DD *
 DISPLAY FROM(IN1) LIST(REPORT) DATE  TIME PAGE        -
       TITLE('         SYSTEM DESIGN CHECK         ')  -
      HEADER('DATASET NAME ')    ON(Ø1,44,CH)          -
      HEADER(' SMP/E DDDEF ')    ON(45,Ø8,CH)          -
      HEADER(' SMP/E ZONES ')    ON(54,16,CH)          -
      HEADER('INDIRECT/CAT ')    ON(7Ø,Ø5,CH)          -
      HEADER(' CATALOGUED ')     ON(76,Ø5,CH)          -
      BLANK   LINES(63)
/*
```

PRINTOUT RESULT

```
Ø1/28/Ø3        13:39:23        - 1 -               SYSTEM DESIGN CHECK

 DATASET NAME       SMP/E DDDEF  SMP/E ZONES   INDIRECT/CAT   CATALOGUED
 ---------------    ----------- ----------    ------------   ----------

 AOP.SAOPEXEC       SAOPEXE      TARGET_zone   YES             YES
 AOP.SAOPMENU       SAOPMEN      TARGET_zone   YES             YES
 AOP.SAOPPENU       SAOPPEN      TARGET_zone   YES             YES
```

```
.............................................................
GLD.SGLDLNK.COPY                  OPERATIONAL  NO             NO
GSK.SGSKLOAD       SGSKLOA      TARGET_zone   YES            YES
ICA.SICALMOD       SICALMO      TARGET_zone   YES            YES
..............................................................
IOE.SIOEPROC       SIOEPRO      TARGET_zone   YES            YES
ISF.SISFEXEC       SISFEXE      TARGET_zone   YES            YES
ISF.SISFLINK       SISFLIN      TARGET_zone   YES            YES
ISF.SISFLOAD       SISFLOA      TARGET_zone   YES            YES
..............................................................
SYS1.ADGTPLIB      ADGTPLI      DISTRIBUTION_zone    NO      YES
SYS1.AERBPWSV      AERBPWS      DISTRIBUTION_zone    YES     YES
SYS1.PARMLIB                    OPERATIONAL          NO      YES
...............................................................
SYS1.VTAMLST                    OPERATIONAL   YES            YES
TCPIP.SEZACMTX     SEZACMT      TARGET_zone   YES            YES
...............................................................
```

Massimo Ambrosini (Italy) © Xephon 2003

Monitoring DFSMS volume selection failure

PROBLEM ADDRESSED

One of the keys to successful storage management is to automate or streamline as many of the routine storage management tasks as possible, leaving the people in an organization free to be more productive in their own job assignments. This task of storage management is basically what DFSMS is responsible for.

However, every now and then it happens that some strange message appears notifying the user of volume selection failure because of insufficient space when allocating a dataset. The problem is that these messages are appearing only in the user's job log, so there is no way to automatically trap them or to do anything about it. To make it a bit more complicated, there is in fact a set of DFSMS messages dealing with volume selection failure and each one is worth taking into consideration.

A partial list of these messages looks like this:

- IGD17206I VOLUME SELECTION HAS FAILED - THERE ARE NOT ENOUGH VOLUMES WITH SUFFICIENT SPACE FOR DATA SET dsname.
- IGD17207I VOLUME SELECTION HAS FAILED - THERE ARE NO ACCESSIBLE VOLUMES FOR DATA SET dsname.
- IGD17272I VOLUME SELECTION HAS FAILED FOR INSUFFICIENT SPACE FOR DATA SET dsn.
- IGD17273I ALLOCATION HAS FAILED FOR ALL VOLUMES SELECTED FOR DATA SET dsname.
- IGD17274I VOLUMES SPECIFIED FOR A GUARANTEED SPACE REQUEST DO NOT BELONG TO AN ELIGIBLE STORAGE GROUP ALLOCATION FOR DATA SET dsname FAILED.
- IGD17275I NO ELIGIBLE STORAGE GROUP HAS ENOUGH SPACE FOR BEST FIT REQUEST - ALLOCATION FOR DATA SET dsname FAILED.
- IGD17803I VOLUME SELECTION HAS FAILED. THERE ARE ACCESSIBLE VOLUME(S) BUT NOT ENOUGH WITH SUFFICIENT SPACE FOR DATA SET dsname.
- IGD17804I VOLUME SELECTION HAS FAILED. THERE ARE NO ACCESSIBLE VOLUMES FOR DATA SET dsname.

A flowchart of DFSMS volume selection for a dataset allocation can be found in Chapter 6 of the *DFSMSdfp Storage Administration Guide* (SC26-7402) along with the possible reasons for a volume selection failure.

What was needed was a procedure that would take a snapshot of the VTOCs of the volumes if a storage shortage error occurs and provide an accurate storage group space map.

SOLUTION PROPOSED

A quick and simple way to get the DFSMS storage group report with a summary of free space in a storage group at the moment of volume allocation failure is available from MXI (MVS eXtended Information). MXI is free and available from Scott Enterprise Consultancy (http://www.secltd.co.uk). It does not use any method of CPU serial number protection or encryption. No passwords or activation zaps are required.

A neat REXX interface that MXI provides was utilized to get the information needed. The following REXX EXEC (MXISGP) lists the storage groups by using the SGRP command.

MXISGP EXEC

```
/* REXX - EXEC to invoke MXI Storage Group Report   */
/* using LIBDEFs                                    */
/*                                                  */
arg hlq
if hlq = "" then HLQ = 'SYS2.MXI'
address ISPEXEC "LIBDEF ISPTLIB DATASET ID('"hlq".TABLES') STACK"
address ISPEXEC "LIBDEF ISPLLIB DATASET ID('"hlq".LOAD') STACK"
   say ''
   say " Storage Group Report for System" MVSVAR(SYSNAME),
        ||" running" MVSVAR(SYSOPSYS)
   say ''
   say ''left(' ',1Ø,' '),
         ||center('Report produced on',18,),
         ||left(' ',1,' ')||left(date(),11),
         ||left(' ',1,' ')||left('at ',3,' '),
         ||left(time(),1Ø)
   say ''
  X = MXIREXX('LINE.','SGRP')
  DO I = 2 TO LINE.Ø
    X = STRIP(LINE.I)
    IF X <> ''
    THEN SAY LEFT(LINE.I,78)
  END
ADDRESS ISPEXEC "LIBDEF ISPLLIB";
ADDRESS ISPEXEC "LIBDEF ISPTLIB";
exit
```

The above EXEC gives the following output (without the headings!):

```
Name    Type   Total   Free   %Use   Hi Low     Freq PSM IMG AMG BKP DMP
SGTEST  POOL   1Ø828     714Ø   34   85   Ø -------- No  No  No  No  No
SGBKP1  POOL   165127  1Ø4232   36   1Ø   5  NoLimit No  IMG AMG No  No
SGBKP2  POOL   1Ø828Ø   622Ø8   42   1Ø   5  NoLimit No  IMG AMG No  No
```

Output fields legend:

- Name – storage group name.
- Type – the type of storage group (POOL/VIO/TAPE/OBJECT/OBJ-BACK).
- Total – total space (in MB) in the storage group.
- Free – free space (in MB) available in the storage group.
- %Use – used space percentage.
- Hi – migration threshold (high).
- Low – migration threshold (low).
- Freq – back-up frequency.
- PSM – is Primary Space Management active (PSM/No).
- IMG – is Interval Migration active (IMG/No).
- AMG – is Automatic Migration active (AMG/No).
- BKP – is Automatic Backup active (BKP/No)
- DMP – is Automatic Dump active (DMP/No)

The space usage figures reported for storage groups are obtained from DFSMS, via a subsystem call, by asking for the VLD information (mapped by IGDVLD in SYS1.MODGEN). Getting the VLD information for the storage group is much faster than the more commonly used LISTVOL in ISMF, since ISMF goes to the VTOC for each volume within the storage group to get the free space information and thus could be impacted by RESERVEs.

While being an excellent tool for taking an instant snapshot of storage group space figures at the moment of volume selection failure, MXI may not produce the most accurate and up-to-date space usage information for a storage group. Therefore, a batch procedure was constructed with the aim of producing a more precise storage group space report. The procedure should be submitted as soon as the DFSMS allocation failure occurs in order to get a valid snapshot of the storage groups.

The procedure is a two-part stream – in the first part (DCOLL) DCOLLECT was used to collect on the storage groups data. In the second part (STORG) the captured records are formatted and reports produced. We did not need a fancy reporting program to extract a useful report from the DCOLLECT output file. We just used the field reformatting capability of DFSORT's ICETOOL to produce sorted reports from the DCOLLECT output.

The first report (Storage Group Summary Space Report) can give us an idea of how the space in the storage groups is being utilized. It provides us with the number of volumes in SG, allocated space, free space in SG, number of free extents in SG, the size of the largest extent in SG, total number of free DSCBs, and total number of free VIRS. The second report (Detailed Storage Group Space Report) provides additional information at the volume level.

```
//DEL       EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=X
//SYSIN     DD *
    DELETE uid.DCOLL.OUTPUT
    SET MAXCC=Ø
/*
//DCOLL    EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=X
//OUTDS    DD DSN=uid..DCOLL.OUTPUT,DSORG=PS,
//         DCB=(RECFM=VB,LRECL=644,BLKSIZE=Ø),
//         SPACE=(1,(1ØØ,1ØØ)),AVGREC=K,
//         DISP=(NEW,CATLG,KEEP)
//SYSIN    DD     *
       SET MAXCC=Ø
       DCOLLECT -
       OFILE(OUTDS) -
       STORAGEGROUP(storage group1  -
                    storage group2  -
                    ….
                    storage groupn) -
       NODATAINFO
/*
//STORG   EXEC  PGM=ICETOOL,REGION=1Ø24K
//DFSMSG    DD  SYSOUT=*
//TOOLMSG   DD  SYSOUT=*
//INDD      DD  DSN=uid.DCOLL.OUTPUT,DISP=SHR
//OUTDD1    DD  DSN=&&TEMP1,DISP=(NEW,PASS),LIKE=uid.DCOLL.OUTPUT
//OUTDD2    DD  DSN=&&TEMP2,DISP=(NEW,PASS),LIKE=uid.DCOLL.OUTPUT
//SGSUMM    DD  SYSOUT=*
```

```
//SGRPT     DD  SYSOUT=*
//TOOLIN    DD  *
*  Create a temporary dataset with 'V' type records only
   COPY FROM(INDD) TO(OUTDD1) USING(CPY1)
*
    OCCUR FROM(OUTDD1) LIST(SGSUMM) -
    TITLE('Dcollect output Records produced on:') -
    HEADER('SID')      ON(13,4,CH)             -
    HEADER('Date')     ON(21,4,DT1,E'9999/99/99')-
    HEADER('Time')     ON(17,4,TM1,E'99:99:99')  -
    BLANK
*
   SORT FROM(OUTDD1) TO(OUTDD2) USING(SRT1)
*
*  Print a report showing total allocated / free space by SG
*
  DISPLAY FROM(OUTDD2) LIST(SGRPT)  -
  TITLE('Datailed Storage Group Space Report (in MB)') -
  HEADER('Volume')       ON(29,6,CH)      -
  HEADER('Allocated')    ON(45,4,BI,/KB)  -
  HEADER('Free')         ON(41,4,BI,/KB)  -
  HEADER('Free Ext.')    ON(61,4,FI)      -
  HEADER('Largest Ext.') ON(57,4,BI,/KB)  -
  HEADER('Free DSCB')    ON(65,4,FI)      -
  HEADER('Free VIRS')    ON(69,4,FI)      -
  BREAK(87,1Ø,CH)                         -
  BTITLE('Group Report')                  -
  BLANK
*
//CPY1CNTL DD *
* Include V-type records only
 INCLUDE COND=(9,1,CH,EQ,C'V')
* Create report
 OUTFIL FNAMES=SGSUMM,
   NODETAIL,
   HEADER1=(1:'Storage Group Summary Space Report (in KB)',/,
            1:1ØX,/,
            1:'Group',11:'# Volumes',28:'Alloc ',
            42:'Free',49:'Free Ext',
            6Ø:'Large Ext',71:'Free DSCB',82:'Free VIRS'/,
            1:'------',11:'--------',25:'--------',
            39:'------',49:'--------',6Ø:'--------',
            7Ø:' --------',81:' --------'),
     SECTIONS=(87,1Ø,
       TRAILER3=(1:87,1Ø,           * Storage Group
       12:COUNT,                    * no.of Volumes in SG
       23:TOT=(45,4,BI),            * Allocated Space in SG
       36:TOT=(41,4,BI),            * Free Space in SG
       47:TOT=(61,4,FI),            * number of Free Extents in SG
       59:MAX=(57,4,BI),            * Largest Ext. in SG
       7Ø:TOT=(65,4,FI),            * Total no. of Free DSCBs
       81:TOT=(69,4,FI)))           * Total no. of free VIRS
/*
//SRT1CNTL DD *
* Sort by Storage group, Volume serial
 SORT FIELDS=(87,1Ø,CH,A,29,6,CH,A)
/*
```

It should be noted that this procedure may be executed on an interval basis, say every hour, so that these reports can give us an idea of what is going on and thus enable a storage administrator to manage and monitor storage groups more effectively. For example, someone could allocate and then delete a large dataset in the hour between the two samples being taken – this would not be reflected in the data. But taken over a month this data is likely to give us a very good idea of what is going on, ie we can get an idea of what the allocation is like at noon on an average day, or we can get an idea of what the maximum allocation at noon was during the month.

Worthy of the storage administrator's attention is the fact that each time volume selection failure occurs, SMF record type 42 subtype 10 (introduced with z/OS V1R3) is written. Based on that fact, the second procedure was created to enable us to keep track of DFSMS volume selection failure and tune DFSMS.

Again, the procedure is a two-part stream – in the first part (SMF42), selected SMF records are dumped from the SMF dataset to a file, which can be used as a base of archived records. In the second part (RPT42A), the captured records are formatted and a report produced. The report shows the job and user requesting the dataset allocation that failed, along with the amount of space requested, dataset name, storage group that failed, as well as the date and time allocation failure took place.

```
//DEL       EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=X
//SYSIN     DD *
     DELETE uid.SMF42.OUT
     DELETE uid.SMF42A.OUT
     SET MAXCC=Ø
/*
//SMF42    EXEC PGM=IFASMFDP,REGION=5M
//INDA1 DD DSN=xxx.SMFDUMPW,DISP=SHR               * WEEKLY SMF DS
```

```
//*NDA2 DD DSN=SYS1.MANx,DISP=SHR,AMP='BUFND=151'  * OR CURRENT SMF DS
//*NDA3 DD DSN=SYS1.MANy,DISP=SHR,AMP='BUFND=151'
//OUTDA DD DSN=uid.SMF42.OUT,DISP=(NEW,CATLG),UNIT=SYSDA,
//         SPACE=(CYL,(5Ø,2Ø),RLSE)
//*        DCB=(xxx.SMFDUMPW)
//SYSPRINT DD SYSOUT=X
//SYSIN DD *
       INDD(INDA1,OPTIONS(DUMP))
       OUTDD(OUTDA,TYPE(42))
/*
//SMF421Ø  EXEC PGM=ICETOOL
//TOOLMSG  DD SYSOUT=*
//DFSMSG   DD SYSOUT=*
//SMF42    DD DSN=uid.SMF42.OUT,DISP=SHR
//SMF421Ø  DD DSN=uid.SMF42A.OUT,
//         SPACE=(CYL,(9Ø,9Ø)),UNIT=SYSDA,
//         DISP=(NEW,CATLG,DELETE),
//         DCB=(RECFM=VB,LRECL=32756,BLKSIZE=3276Ø)
//TOOLIN   DD *
  COPY FROM(SMF42) TO(SMF421Ø) USING(SMFI)
//SMFICNTL DD *
  OPTION SPANINC=RC4,VLSHRT
  INCLUDE COND=(6,1,BI,EQ,42,AND,23,2,BI,EQ,1Ø)
/*
//RPT42A   EXEC PGM=ICETOOL
//TOOLMSG  DD SYSOUT=*
//DFSMSG   DD SYSOUT=*
//SMF421Ø  DD DSN=uid.SMF42A.OUT,DISP=SHR
//OUTDD2   DD DSN=&&TEMP1,DISP=(NEW,PASS),
//         SPACE=(CYL,(9Ø,9Ø)),UNIT=SYSDA
//SMFREP   DD SYSOUT=*
//SGRPT    DD SYSOUT=*
//TOOLIN   DD *
  DISPLAY FROM(SMF421Ø) LIST(SMFREP) -
  TITLE('DFSMS Volume Selection Failure') DATE(4MD/)  TIME -
  HEADER('SID')           ON(15,4,CH)                 -
  HEADER('Date')          ON(11,4,DT1,E'9999/99/99') -
  HEADER('Time')          ON(7,4,TM1,E'99:99:99')    -
  HEADER('Job Name')      ON(85,8,CH)                 -
  HEADER('Pgm Name')      ON(93,8,CH)                 -
  HEADER('DD name')       ON(1Ø9,8,CH)                -
  HEADER('Step Name')     ON(1Ø1,8,CH)                -
  HEADER('Req. space')    ON(161,4,BI)                -
  HEADER('Unt')           ON(165,4,CH)                -
  HEADER('Storage Gr.')   ON(26Ø,12,CH)               -
  HEADER('Dataset')       ON(117,44,CH)               -
  HEADER('Data Class')    ON(167,1Ø,CH)               -
  HEADER('Mgt. Class')    ON(199,1Ø,CH)               -
  HEADER('Storage Class') ON(229,11,CH)               -
  BLANK
```

```
*
*  Create a temporary dataset containing dataset records that
*  are sorted by Job name, with the allocated space totalled
*  (SUMmed)
*
  SORT FROM(SMF421Ø) TO(OUTDD2) USING(SRT1)
*
*  Print a report showing total space requested by Date
*  including the maximum, minimum, and average space requested
*
 DISPLAY FROM(OUTDD2) LIST(SGRPT) BLANK -
  TITLE('Total Space Requested (MB) by Date') DATE TIME -
   HEADER('Date')            ON(11,4,DT1,E'9999/99/99') -
   HEADER('Total Space (MB)') ON(161,4,BI,/KB) -
   MAXIMUM('Most Space:')                     -
   MINIMUM('Least Space:')                    -
   AVERAGE('Average Space:')
*
  SORT FROM(SMF421Ø) TO(OUTDD2) USING(SRT2)
*
 DISPLAY FROM(OUTDD2) LIST(SGRPT) BLANK -
  TITLE('Total Space Requested (MB) by SG') DATE TIME -
   HEADER('Storage Gr.')       ON(26Ø,12,CH)    -
   HEADER('Total Space (MB)')  ON(161,4,BI,/KB) -
   MAXIMUM('Most Space:') -
   MINIMUM('Least Space:') -
   AVERAGE('Average Space:')
*
  SORT FROM(SMF421Ø) TO(OUTDD2) USING(SRT3)
*
 DISPLAY FROM(OUTDD2) LIST(SGRPT) BLANK -
  TITLE('Total Space Requested (MB) by Job') DATE TIME -
   HEADER('Job')              ON(85,8,CH)  -
   HEADER('Total Space (MB)') ON(161,4,BI,/KB) -
   HEADER('Storage Gr.')      ON(26Ø,12,CH)    -
   MAXIMUM('Most Space:') -
   MINIMUM('Least Space:') -
   AVERAGE('Average Space:')
/*
//SRT1CNTL DD *
* sort by Date and sum (total) space requested
 SORT FIELDS=(11,4,BI,A)
 SUM FIELDS=(161,4,BI)
/*
//SRT2CNTL DD *
* sort by SG and sum (total) space requested
 SORT FIELDS=(26Ø,12,CH,D)
 SUM FIELDS=(161,4,BI)
/*
//SRT3CNTL DD *
```

```
* sort by Job and sum (total) space requested
 SORT FIELDS=(85,8,CH,A)
 SUM FIELDS=(161,4,BI)
/*
```

Mile Pekic
Systems Programmer (Serbia and Montenegro)

CPSM/SPOC performance study

EXECUTIVE SUMMARY

The results of the CPSM/SPOC performance study proved that it is possible to create a CPSM monitoring environment that will satisfy all operational monitoring objectives without jeopardizing real storage.

CPSM monitoring will be limited to the CPSM Web User Interface (WUI). We will create customized menus and views that completely eliminate the possibility of an innocent selection of any storage intensive views.

This testing process was intensive and time-consuming. Additional CPSM internals documentation from IBM could have circumvented the need to expend the resources to collect, analyse, and deduce processing behaviour.

Until CPSM internals documentation exists, we recommend similar performance testing to measure storage usage prior to implementing any additional phases of the CPSM project.

PROBLEM

Identify 'storage friendly' query techniques to facilitate a successful, Single Point of Control (SPOC) operational monitoring implementation of CPSM in a large storage-constrained environment.

BACKGROUND

Our CPSM implementation has been attempted several times over the last 26 months. The initial attempts were thwarted by a CPSM design issue that was not uncovered until a customer attempted to implement the product. The re-introduction of the CPSM project with a new code base has been much more carefully managed. Unfortunately, the most recent CPSM implementation exposed adverse storage management behaviour that allowed uncontrollable real storage usage during operational monitoring.

After joint discussions with IBM regarding possible programming solutions and an agreement on some long-term approaches, a short-term interim solution was recommended. This approach suggested removing general access to all CPSM user interfaces except the Web User Interface. We would then tailor the product's menus and views by using the standard customization dialogs to remove access to the 'offending' views.

Monitoring techniques vary between the ISPF End User Interface (EUI), the Web User Interface (WUI), and the batch REXX/API. The SPOC phase of the project is eagerly anticipated by operations to gain new insight into a complex difficult-to-monitor CICS environment.

Existing monitoring requirements dictate that predefined proactive 'health checks' must be run periodically throughout the day and when a problem is suspected. These involve activities that proactively look for CICS resources that have changed from the desired state, but may not have tripped an Omegamon exception. These include, but are not limited to:

- Disabled programs (PROGRAM).
- Disabled transactions (LOCTRAN/REMTRAN).
- Released connections (CONNECT).
- Out-of-service terminals and printers (TERMNL).
- Creeping DB2 thread use count (DB2THD).

Real-time monitoring still includes Omegaview with Omegamon/ CICS but has been augmented with CPSM for improved visibility to:

- Long-running UOWs (30 seconds to 10 minutes; filtered UOWORK).
- All non-overhead active tasks (filtered TASK).
- Sysdumps (Sorted SYSDUMP).
- Trandumps (Sorted TRANDUMP).

During the most recent implementation attempt, several occurrences of extreme storage bloat occurred while accessing many of the higher-volume CPSM Resource Tables. In the extreme cases this caused an IPL of the host LPAR.

IBM's recommendation was to procedurally force all users to specify a 'scope' on all views to avoid the problem. Our monitoring strategy depends on monitoring at the CICSPLEX level (SCOPE=CICSPLEX). Even though we have a fairly sophisticated set of CICSGRP definitions, the 'scope' limitation would severely limit the value of CPSM for SPOC and is an unacceptable solution. The product's basic menu navigation in both the EUI and WUI promote innocent usage of the 'offending views'. Our position is that the product should not be the cause of catastrophic storage shortages nor should it restrict us from accessing any piece of CPSM-related data.

Since the default IBM Menu/View Starter Set allows access to everything, IBM again recommended building menus and views with 'restricted' access. Again, the recommendation was based on the unacceptable approach of using 'scoped' views.

Internal testing with the EUI and the API tended to suggest that it was possible to formulate queries that did not force the use of 'scopes' but still managed the CPSM storage more efficiently. Unfortunately, IBM does not document the internals of CPSM and would not answer direct questions about the storage management techniques used during CPSM query processing.

HYPOTHESIS

CPSM queries using ‘filters’ can be more effective from an operational perspective and a storage management perspective than ‘scopes’.

ENVIRONMENT

The production environment encompasses three large parallel sysplexes. Sysplex 1 is the primary processing environment with 12 LPARs on six CECs supporting 160 CICS regions (seven TORs, 127 AORs ten QORs, six MQ AORs, ten RORs) and 10-way DB2 Data Sharing. Sysplex 2 is the customer facing Web environment with four LPARs on two CECs supporting 50 CICS Regions (32 AORs, 16 MQ AORs, two RORs) and 4-way DB2 Data Sharing. Sysplex 3 is a dedicated ‘hot spare’ back-up and maintenance environment with three LPARs on two CECs supporting 43 CICS regions (seven TORs, 34 AORs, two RORs) and 2-way DB2 Data Sharing.

Sysplexes 1 and 3 make up CICSPLEX 1 and for disaster recovery/isolation reasons Sysplex 2 is CICSPLEX 2. CICSPLEX 1 is supported by 14 CMASs (12 routing CMASs and two hub CMASs). CICSPLEX2 is supported by five CMASs (four routing CMASs and one hub CMAS). Both Sysplexes in CICSPLEX 1 are running OS/390 V2R10 and are currently still using 31-bit architecture. The Sysplex in CICSPLEX 2 is running OS/390 V2R10 using 64-bit architecture.

CICSPLEX 1 has the following volumes in the CPSM Resource Tables used in the test queries:

- CICSRGN – 151
- CONNECT – 57,064
- PROGRAM – 1,736,080.

A test CICPlex was also used to test the extreme case since this was not desirable in the production environment. The test Sysplex is two LPARs on two shared CECs with 114 CICS regions. The test Sysplex is running OS/390 V2R10 and 64-bit architecture:

- CICSRGN–114
- CONNECT–8,600
- PROGRAM–1,004,000.

TOOLS

The toolset for this performance testing was as follows.

Omegamon/MVS

The Candle Omegamon/MVS monitor was used to watch the CPU utilization and Working Set Size (WSS) of address spaces targeted for testing.

CA/Sysview

The Computer Associates Sysview DSLIST feature of Sysview was used to monitor dataspace frame growth at the individual dataspace level. This was the only tool we could find that displayed individual dataspace frame count usage.

RMF

The RMF Monitor III was used to watch for delays and to review one- minute historical snapshots of the targeted address spaces. The RMF post processor was used to process SMF type 79 records to produce a timeline showing the WSS of the target address spaces during the test.

SDSF

SDSF was used as a real-time indicator for cross systems activity. An effective naming convention allows the monitoring of several address spaces across several LPARs simultaneously. All CAS, CMAS, and WUI address spaces were monitored for CPU%, Real Storage Frame Counts, SIO, and EXCPs. The ESSS address space was watched on a separate screen.

Homegrown CPSMRW REXX EXEC

A REXX EXEC was written to allow generic report writer activities against all CPSM data. This EXEC provides quick access to all CPSM Resource Tables' contents. It also provides the ability to quickly define report formats, set FILTERs, perform GROUPing summaries, and establish ORDERs using simple PARMLIB members and JCL. CPSMRW also writes diagnostic messages as it completes each phase of execution. It logs when the Connect occurs, it logs when the GETDEF completes and provides the object length, it logs when the GET completes, it logs when the GROUP BY completes, it logs when ORDER BY completes, it logs when all FETCHs complete, and finally it logs when the Disconnect occurs. Aside from providing an easy-to-use CPSM Report Writer, this allows us to see where the queries are spending most of their time in a known processing sequence.

PROCEDURE

The basic approach is to use the toolset to monitor CPU utilization and WSS growth for all targeted address spaces at specific checkpoints during the test script. Since we know the processing sequence of the batch REXX/API jobs, we can compare monitor results from a known request with the monitor results of a similar WUI request. The hope is to learn basic API processing behaviour and by association learn WUI processing behaviour.

The procedures involved identifying a set of representative queries that can be executed using the batch REXX/API, then run the same query again using the WUI. Since the WUI defaults to executing unscoped and unfiltered views, we use tailored URLs to launch all WUI queries.

The address spaces monitored during the test were:

- CAS
- CMAS
- ESSS

- WUI
- Batch REXX/API job.

A baseline for all address spaces was taken before the test began. The basic measurements for each address space were:

- CPU Utilization
- WSS
- Used Frame Count for each dataspace for each address space.

After the baseline was taken, all CICS regions were brought down and all CMASs and CASs were brought down. It was confirmed at this point that the ESSS released all the acquired dataspace storage frames.

CASs and CMASs were brought back up and measurements were taken without any CICS regions. Next all the CICS regions were brought up. After all regions received CONTROL GIVEN and the TOP1xxxx dataspace stopped growing, another set of measurements was taken.

TEST SCRIPT

The following queries were run and measurements were taken as each query was in progress and as each query completed:

1. Simple GET ALL/FETCH ALL CICSRGN object using the API.
2. http://host:6999/cicsplexsm//userid/view/eyustartcicsrgn.
3. GET (STATUS=DISABLED.)/FETCH ALL PROGRAM object using the API.
4. http://host:6999/cicsplexsm//userid/view/eyustartprogram?a_status=disabled.
5. http://host:6999/cicsplexsm//userid/view/eyustartconnect.

6 Simple GET ALL/FETCH ALL CONNECT object using the API.

7 http://host:6999/cicsplexsm//userid/view/eyustartprogram.

8 Simple GET ALL/FETCH ALL PROGRAM object using the API.

Starting at test 5 we flipped the order and performed the WUI test first and the API test second.

OBSERVATIONS

The following observations are placed up front to simplify the presentation of the measurements:

- The CAS did not actively participate in any of the tests and therefore did not register any significant CPU utilization or WSS growth.
- The batch REXX/API address spaces did not register any significant CPU utilization or WSS growth. In larger views, the only noticeable characteristic was the FETCH/SAY processing loop. Using the CPSMRW diagnostic messages it was possible to correlate the CPU/Real/SIO/EXCP activity with the report generation process. This accounted for all the CPU and EXCP processing since the API jobs appeared to be swapped out during all CPSM GET processing.
- The WUI regions did not register any significant CPU utilization or WSS growth. This region exhibited all the characteristics of a normal CICS region.
- The CMAS registered the majority of the CPU utilization on behalf of all requests (WUI or batch REXX/API).
- None of the tests suggested that the CMAS was processor-bound or using a disproportionate amount of CPU. All CMAS CPU Utilization measurements were in a very reasonable non-noteworthy range.
- The CMAS did not register any significant WSS growth.

- The ESSS did not register any significant CPU utilization.
- The ESSS was the significant contributor to all WSS growth.
- The ESSS is the largest user of dataspaces in the system.

ADDITIONAL GENERAL OBSERVATIONS

We had already heard that the ESSS would not return storage once acquired because of built in high-water mark processing. In anticipation of this behaviour we sequenced our tests in an order that we believed would allow us to observe a constantly growing WSS. We did observe that once storage was acquired by the ESSS for a dataspace it was never returned.

During the start-up of the CMASs, all but three of the dataspaces grew to static sizes and were unchanged throughout the remainder of the tests. The three dataspaces that were influenced by the CICS regions were TOP1xxxx, QUE1xxxx, and DAT1xxxx.

As each region joined the CICSPlex and completed LMAS initialization, we saw the TOP1xxxx dataspace grow. Once all regions were active, the TOP1xxxx dataspace did not experience any further growth.

All storage impacts of query processing were localized to the ESSS on the LPAR hosting the query (either the batch REXX/API job or the WUI). We did not observe any cross-plex storage impacts during CPSM Query processing.

The QUE1xxxx dataspace appears to grow during CPSM GET processing. The DAT1xxxx dataspace appears to grow either at the conclusion of the GET processing or as soon as FETCH processing begins.

There appears to be about 32% overhead in the QUE1xxxx and DAT1xxxx dataspaces. If these dataspaces are holding the full contents of each CPSM Resource Table row then the calculation of row length times number of rows comes out to slightly better than 68% of the actual storage usage of the dataspaces.

The WUI appears to be more efficient at returning results and

	Baseline	Down	CMAS Up	CICS Up	API 1	WUI 1	API 2	WUI 2	WUI 3	API 3	WUI 4	API 4
WSS	518M	Ø	82M	271M	271M	272M	272M	272M	277M	281M	1548M	2179M
DAT	6M	Ø	4M	4M	4M	4M	5M	5M	8M	12M	671M	133ØM
QUE	31M	Ø	2M	9M	9M	9M	9M	9M	1ØM	1ØM	682M	682M
TOP	121M	Ø	1M	97M	97M	97M	97M	97M	97M	97M	97M	97M

Figure 1: CPSM, ESSS, WSS, and dataspace frame growth in bytes

control to the user than API programs that attempt to FETCH through all results before returning the results set to the user. The WUI seems to FETCH only enough data to fill the generated Web page with a preset number of rows. This preset number of rows varies by view, but seems to be around 20 on many views. This suggests that the WUI will hold on to the CPSM connection longer while the user ponders the results and pages forward through the results. This also suggests that the contents of the DAT1xxxx dataspace are still in use and those dataspace storage frames cannot be reused until the query is terminated. Assuming this is correct, we believe we saw confirmation that CPSM does not reuse result sets when simultaneous requests for the same data occur. We saw the DAT1xxxx dataspace double in size during our last test when the WUI still displayed the first page of the extremely large result set from the unfiltered PROGRAM view and we launched the API query for the same data.

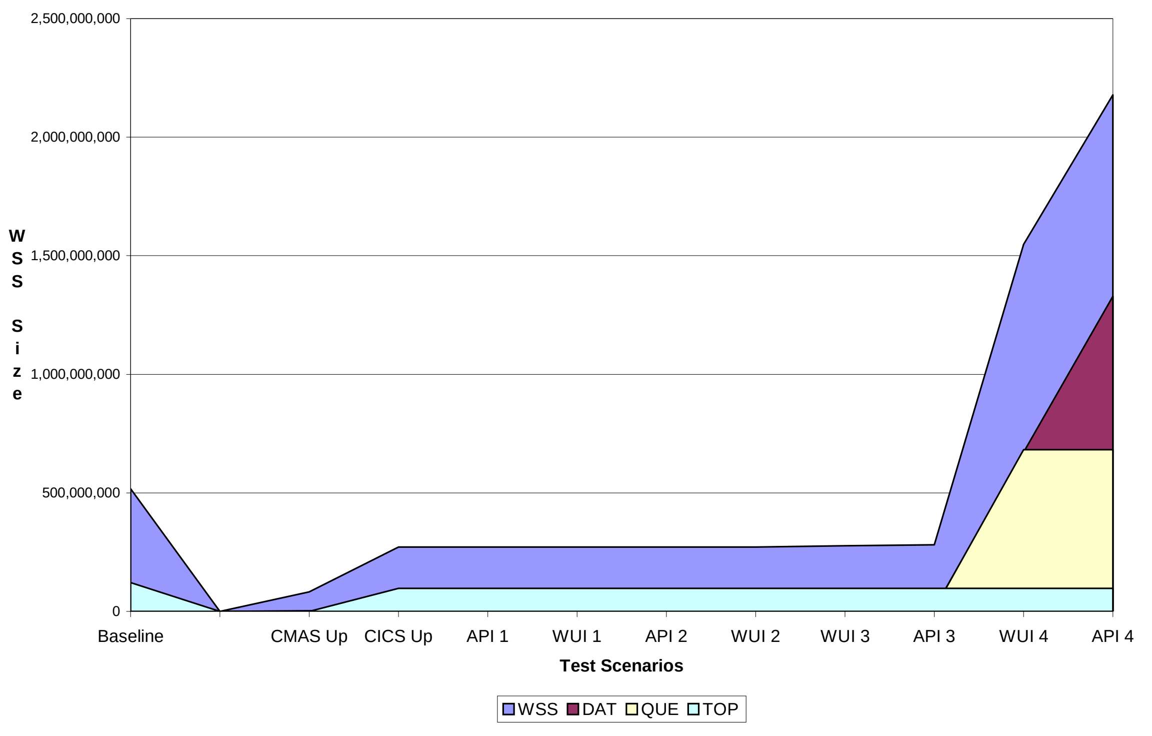

Figure 2: Graph of CPSM, ESSS, WSS, and dataspace frame growth in bytes

We observed a slight jump in CICS CPU utilization during higher volume and larger object queries. When a PROGRAM query using the STATUS=DISABLED filter was run using the API or the WUI we saw all the CICS regions 'light up'. It appears the regions are polled for this data as opposed to satisfying the query from the topology data in the TOP1xxxx dataspace.

Example data is shown in Figures 1 and 2.

CONCLUSIONS

It is possible to use storage-friendly query techniques to facilitate a successful, Single Point of Control (SPOC) operational monitoring implementation of CPSM in a large storage-constrained environment.

Armed with the knowledge that CPSM filters provide us with the mechanism to calculate and control the maximum CPSM query storage impact, we can devise an implementation strategy that will guarantee that CPSM will never run away with all the storage while satisfying the operational requirements. This strategy will be based on the IBM recommendation of implementing a tailored set of WUI menus and views with a known maximum storage impact. This will be combined with an effective use of the WUI MAXUSER parameter to complete the storage management controls.

After observing several undocumented storage behaviours, it is advisable to formulate a similar test to learn the storage usage patterns prior to implementing any new features in CPSM.

We would like IBM to review this report and confirm the conclusions or give us feedback on where we may have misinterpreted the data.

FOLLOW-ON RESEARCH TOPICS

During the research, other questions and follow-on projects were noted. In some cases, these questions will be posed to IBM. In others, these are notes for future testing prior to implementing any additional phases of CPSM.

We speculated that setting the WUI row count per Web page higher would reduce the likelihood of a CPSM connection hold ing DAT1xxxx dataspace storage frames, but could inflict a DSA problem on the WUI region.

Since there is no documentation on the ESSS dataspaces and which components use them and under what circumstances, future tests are advisable prior to implementing any additional CPSM features. If it were possible to get documentation on the dataspaces and their uses from IBM, we could avoid this time-consuming testing process.

After observing the storage behaviours in the product, we would recommend some additional controls in the product. The purpose of these controls would be to set storage objectives for the CPSM product. We would find great value in the ability to specify values for a maximum amount of dataspace storage as well as a minimum amount to allow efficient contraction of dataspace storage. Another set of controls that would provide a high degree of value would be user limits and storage warnings when queries exceed predefined thresholds.

ADDITIONAL NOTES

Since this report was initially written there have been many developments to enhance our understanding of CPSM storage usage, the benefits of an effective WUI implementation, the impact of 64-bit real storage, and z/OS.

The tailored WUI menus and views had a significant impact on CPSM storage usage for operational monitoring. WUI tailoring is a whole other topic, but, suffice it to say, the tailored views are more storage-efficient, easier to use, easier to understand, and directly reflect the monitoring goals of operations. The keys are requirements definition and effective filtering.

The impact of 64-bit real storage was dramatic. This allows all address spaces to use up to 2GB of real storage. Even though our large production plex is still OS/390 2.10, we have tuned many products to put a lot more data in dataspaces. DB2 now puts all its bufferpools in dataspaces. Our DBM1 address spaces are now the biggest, weighing in at over 4GB (address space plus all bufferpool dataspaces). While the ESSS (EYUXnnn) can still get up there, it is not the big kid on the block any more. We have enough real storage allocated to each LPAR (about 20GB) that we can actually handle the unfiltered un-scoped queries alongside everything else. They will still take a while, but it does not consume enough memory to cause us to start thrashing (or worse yet, force an IPL). We still restrict this activity, since the CICS polling that occurs can impact user response time.

The architecture of CPSM does not change with z/OS, therefore the limitation is still physical memory. While our production environments are just beginning to migrate to z/OS, our test z/OS environments can still get storage-contrained. Our test environment is somewhat unrealistic for measurement purposes (114 CICS regions, 20+ DB2s, 20+ Datacoms, 30+ MQ QMGRs, etc). The 64GB of real storage on that CEC can still dwindle under the load. Unfortunately, we still do see a periodic thrashing problem, but at least now we know we can buy more memory to fix it.

Robert Zenuk
Systems Programmer (USA)

Implementing Sub-Capacity Reporting Tool (SCRT) for Workload Licence Charges (WLC)

INTRODUCTION

Prior to z/OS, IBM software products running on OS/390 were typically priced based on the computing capacity of the Central Processor Complex (CPC) on which the software was running.

This pricing method had some limitations:

- If you are running multiple workloads (using IBM licenced software) on the same CPC, you are charged for those products as though each IBM licence is using 100% of the CPC all of the time.
- If you purchase more system capacity (hardware) for an existing CPC, your software charges will increase, even if the added capacity is intended for future growth or for new workloads with different software products.

Because software has become a large part of most IS budgets, IBM tries to give a solution to these problems; this is the goal of Workload Licence Charges (WLC).

With z/OS running on a zSeries server, under WLC you can be charged – for selected IBM products – for less than the full capacity of the CPC on which the products run.

Eligible products fall into one of the following two categories of WLC pricing:

- Variable Workload Licence Charge (VWLC) products. For these products your charges may be based on less than the full capacity of the CPC on which the products run. For the same number of MSU, VWLC products (except z/OS) are priced 10% higher than the PSLC charge. It means that if your CPU usage is higher than 85% of your full capacity, VWLC is not the solution!

- Flat Workload Licence Charge (FWLC) products. These products have a fixed monthly licence charge per product, regardless of the size of the CPC on which the product runs. FWLC products are approximately charged on an 83 MSU basis per CPC.

DB2, CICS, MQSeries and other current middleware products are qualified for VWLC. You can find the list of VWLC eligible products at http://www.ibm.com/zseries/swprice/products.html.

When VWLC products are installed on LPARs using less than the entire capacity of a CPC, WLC allows you to be billed for the VWLC product accordingly. WLC uses the LPAR utilization capacity, which is the highest sum of measured 4-hour rolling MSU averages for the LPARs in which a VWLC product runs concurrently during a given month. For example, if on a Z-900 106 (priced for 199 MSU) with a 100% utilization peak, you run three LPARs with the following capacity and VWLC products balancing:

- LPAR A – 100 MSU – Z/OS + MQ
- LPAR B – 70 MSU. – Z/OS + MQ + CICS + DB2
- LPARC – 29 MSU- z/OS + CICS + DB2.

The VWLC products will be priced based on:

- Z/OS – 100 + 70 + 29 = 199 MSU.
- MQ – 100 + 70 = 170 MSU.
- CICS – 70 + 29 = 79 MSU.
- DB2 – 70 + 29 = 79 MSU.

WLC provides the following advantages:

- Charges for VWLC products are based on how much the LPARs in which the products run utilize system resources, rather than on the full capacity of the CPC.
- You can benefit from ‘white space’ on a CPC. You can divide a CPC into LPARs that have a total utilization less than the

full capacity of the CPC; the portion of hardware that does not carry any software charges is called ‘white space’. ‘White space’ allows you to support more easily short-term workload spikes and long-term workload growth.

- You can implement on the HMC an LPAR ‘defined capacity’, also called a ‘soft cap’. This facility allows the Workload Manager (WLM) to limit the LPAR to no more CPU resource than its defined capacity value. Using ‘defined capacities’, you can control your software bill.

Because IBM is not ready to deliver a production ‘IBM Licence Manager’ (ILM) solution, the ‘Sub-Capacity Reporting Tool’ (SCRT) is an interim solution to implement WLC.

This document describes how to install and implement SCRT.

SCRT IMPLEMENTATION

The Sub-Capacity Reporting Tool (SCRT) is a no-charge IBM tool intended to support sub-capacity software pricing until IBM Licence Manager is available.

SCRT runs as a stand-alone application; it processes System Management Facilities (SMF) records (type 70 – CPU activity/ type 89 – product usage) produced by z/OS and VWLC eligible products.

SCRT analyses a month of data and generates a sub-capacity report that you send to IBM. This report is used to determine your charges for VWLC products.

The output of an SCRT run is a partitioned dataset with a comma-separated-value (Excel .CSV format) member for each CPC containing the sub-capacity report for this CPC.

SCRT installation

SCRT should be downloaded from the zSeries software pricing Web page at http://www.ibm.com/zseries/swprice/scrt.

SCRT is delivered as a ‘load and go’ utility. It means that, in the code source of the JCL downloaded from the Internet, you get the SCRT object module.

```
//SCRTTOOL  JOB (????,????),MSGCLASS=O,NOTIFY=HLQ,REGION=5M
//*
//*  RELEASE 4.2
//*
//*  THIS JOB EXECUTES THE MVS LOADER TO LOAD THE INCLUDED OBJECT
//*  MODULE INTO MEMORY AND EXECUTES IT.  THE PROGRAM WILL READ
//*  AN SMF FILE AND PRODUCE SUB-CAPACITY REPORTING DATA
//*  SUITABLE FOR IMPORT TO YOUR FAVOURITE SPREADSHEET PROGRAM.
//*  AFTER YOU HAVE REVIEWED THE DATA IN THE SPREADSHEET, E-MAIL
//*  THE FILE TO IBM.
//*
//*  DO A GLOBAL CHANGE ON HLQ TO YOUR DESIRED HIGH-LEVEL QUALIFIER.
//*
//*  REPLACE THE JOB STATEMENT WITH ONE THAT FITS YOUR INSTALLATION
//*  STANDARDS.
//*
//*  POINT THE SMF DD AT YOUR SMF DATASET CONTAINING TYPE 7Ø AND 89
//*  RECORDS FOR THE MONTH TO BE REPORTED.  NOTE THAT FOR REPORTING
//*  PURPOSES A MONTH BEGINS AT ØØ:ØØ ON THE SECOND, AND ENDS AT
//*  24:ØØ ON THE FIRST OF THE NEXT MONTH.
//*
//*  UPDATE THE PARMS AND NO 89 DD STATEMENTS BELOW.
//*
//DELETE   EXEC  PGM=IEFBR14
//DD1      DD    DSN=HLQ.CECX.SCRTTOOL.CSV,DISP=(MOD,DELETE),
//             SPACE=(TRK,(1)),UNIT=SYSDA
//SCRT     EXEC  PGM=LOADER
//SYSPRINT DD    SYSOUT=*
//SYSLOUT  DD    SYSOUT=*
//SMF      DD    DISP=SHR,DSN=HLQ.SMF.DATA
//OUTPUT   DD    DISP=(,CATLG),DSN=HLQ.CECX.SCRTTOOL.CSV,
//             SPACE=(TRK,(15,15,15)),UNIT=SYSDA
//PARMS    DD    *
*
* THERE ARE 6 PARM LINES: THE FIRST IS REQUIRED
*
* ENTER YOUR CUSTOMER NAME

*
* ENTER YOUR IBM CUSTOMER NUMBER
55823Ø1
*
* ENTER THE NAME OF THE PERSON SUBMITTING THE REPORT
HARRY FLOWERS
*
* ENTER THE EMAIL ADDRESS OF PERSON SUBMITTING THE REPORT
```

```
HARRY@HARRYSSPORTINGGOODS.COM
*
* ENTER THE PHONE NUMBER OF THE PERSON SUBMITTING THE REPORT
444-555-1212
*
* ENTER THE PO NUMBER
HF8723
*
//NO89      DD  *        NON-89 PRODUCT SECTION
*
* FOR EACH PRODUCT LISTED, YOU MUST ENTER ONE OF THESE 3 OPTIONS:
*
* 1. ENTER *ALL IF THE PRODUCT RUNS IN ALL LPARS IN THE INPUT SMF
*    DATA FILE.
*    E.G.  5655-Ø43=*ALL
* 2. ENTER *NONE IF THE PRODUCT RUNS IN NONE OF THE LPARS IN THE
*    INPUT SMF DATA FILE.
* 3. ENTER A LIST OF LPAR NAMES SEPARATED BY COMMAS OF THE LPARS IN
*    WHICH THE PRODUCT RUNS. (YOU MAY CONTINUE ONTO THE NEXT LINE BY
*    ENDING THE PRIOR LINE WITH A COMMA.)
*
* NOTE: IF YOU HAVE THE SAME LPAR NAME ON DIFFERENT CPCS, AND YOU
*       NEED TO DISTINGUISH BETWEEN THEM, THE FOLLOWING SYNTAX WILL
*       ALLOW IT.  FOR EXAMPLE: YOU HAVE 57XX-XXX RUNNING IN LPARA
*       ON CPC 2Ø64-12345, BUT NOT IN LPARA ON 2Ø64-98765. SPECIFY:
*       57XX-XXX=2Ø64-12345:LPARA
*
* NETVIEW PERF MONITOR V2
5655-Ø43=
* TIVOLI NETVIEW FOR Z/OS
5697-ENV=
* TIVOLI NETVIEW FOR OS/39Ø
5697-B82=
* TIVOLI WORKLOAD SCHEDULER FOR Z/OS
5697-WSZ=
* OPC V2
5697-OPC=
* SYSTEM AUTOMATION FOR OS/39Ø
5645-ØØ5=
* SYSTEM AUTOMATION OS/39Ø V2
5645-ØØ6=
* LOTUS DOMINO FOR S/39Ø
5655-B86=
* COBOL FOR OS/39Ø && VM V2
5648-A25=
* VA PL/I FOR OS/39Ø V2
5655-B22=
* IBM ENTERPRISE PL/I FOR Z/OS AND OS/39Ø V3
5655-H31=
* IBM ENTERPRISE COBOL FOR Z/OS AND OS/39Ø V3
```

```
5655-G53=
* QMF MVS VERSION 3
57Ø6-254=
* DEBUG TOOL FOR Z/OS & OS/39Ø
5655-H32=
//SYSLIN   DD    *

SCRT object module

/*
```

Collecting SMF 70 and 89 records

Before running the SCRT you must collect SMF type 70 and type 89 records for one reporting period.

The following SMFPRM00 statements enable SMF to collect type 70 and type 89 records:

```
SYS(TYPE(7Ø,89))
SUBSYS(STC,TYPE(7Ø,89))
```

Reporting period

IBM billing procedures define a reporting period as the second day of a month to the first day of the next month (for example, from 2 January to the end of the day on 1 February).

SCRT determines the reporting period for a CPC based on the midpoint of the data encountered for that CPC.

A reporting period starts at 00:00 on the second day of one month and continues until 24:00 on the first day of the next month. SCRT discards data from any days outside this reporting period.

The percentage of data collected, a field in the sub-capacity report, is then calculated as actual hours of data divided by expected hours of data. The number of expected hours varies depending on whether a month has 28, 30, or 31 days. Note that if your percentage collected is less than 95%, you must indicate why this is so.

PARMS DD card

This section describes the customer profile: customer name, IBM customer number, etc.

NO89 DD card

You should use this section to enter information about any VWLC products you are using that do not produce SMF type 89 records.

OUTPUT DD card

The OUTPUT DD statement defines the dataset used to store the sub-capacity reports.

This is a PDS with one member for each CPC successfully processed by the SCRT. Each member contains a comma-separated value file (which can be read by any spreadsheet Excel, Lotus, etc) that must be validated by the software asset manager and sent to IBM.

SCRT report

The report produced by SCRT contains several sections:

- Customer information: customer name, customer number.
- Tool information: SCRT release, % of data collected.
- Product summary information: for each product, SCRT computes the WLC utilization capacities.
- Detail LPAR section: for each LPAR, shows when the LPAR reached its maximum MSU utilization.

```
=========== SUB-CAPACITY REPORT==============

Run Date/Time 16 Apr 2ØØ3 - 19:ØØ
Name of Person Submitting Report: zzzz Customer
E-Mail Address of Report Submitter: customer@zzzz.com
Phone Number of Report Submitter: 111-999-9999
Customer Name zzzz Corporation
Customer Number 1234567
Machine Serial Number ØØ-12345
Machine Type and Model 2Ø64-116
Machine Rated Capacity (MSUs) 441
Purchase Order Number 99648
```

```
Is this machine a member of a pricing aggregation? yes
Customer Comments
The data supplied in this sheet will be used to adjust your billing for
all VWLC Programs on this machine. In accordance with our
agreement, IBM will treat a change in product licensed capacity as an
order. If the MSUs have increased since the last report,
your billing will increase. For any VWLC Program, the total number of
MSUs should not exceed the rated machine capacity. Note:
This report is expected to provide a "% data collected" > 95% and data
reporting period beginning on the 2nd of the previous
month and ending on the 1st of the current month.
============================================
TOOL INFORMATION
Tool Release 4.2
Reporting Period 2 Feb, 2ØØ3 - 1 Mar, 2ØØ3
% Data Collected 1ØØ% for 28 days
Any MVS or OS/39Ø executing on this machine? NO <- If yes, ineligible
for sub-CEC
==========================================
PRODUCT SUMMARY INFORMATION
VWLC Product Name VWLC Product ID Tool MSUs CustomerMSUsCustomer
Comments
z/OS V1 5694-AØ1 388 XXXXX runs on SYSA and SYSB
zDB2 for MVS/ESA V4              5695-DB2 Ø    XXXXX
DB2 UDB for OS/39Ø V7            5675-DB2 388  XXXXX    runs on SYSA
and SYSB
DB2 UDB for OS/39Ø V6                 5645-DB2 Ø    XXXXX
CICS TS for OS/39Ø V2                 5697-E93 Ø    XXXXX
CICS TS for OS/39Ø               5655-147 258  XXXXX runs only in SYSA
MQSeries MVS/ESA                 5695-137 Ø    XXXXX
MQSeries for OS/39Ø V5.2              5655-F1Ø Ø    XXXXX
IMS V8                 5655-C56 Ø    XXXXX
IMS V7                 5655-BØ1 162  XXXXX runs only in SYSB
Tivoli NetView for z/OS          5697-ENV Ø    XXXXX
Tivoli Workload Scheduler for z/OS    5697-WSZ Ø    XXXXX
OPC                 5697-OPC Ø    XXXXX
System Automation for OS/39Ø     5645-ØØ5 Ø    XXXXX
Lotus Domino for S/39Ø           5655-B86 388  XXXXX runs in SYSA and
SYSB
COBOL for OS/39Ø & VM V2         5648-A25 Ø    XXXXX
VA PL/I for OS/39Ø          5655-B22 Ø    XXXXX

===================================================================
DETAIL DATA SECTIONS - FOR CUSTOMER ANALYSIS PURPOSES ONLY
===================================================================
DETAIL LPAR DATA SECTION

Highest Hour  Date/Time     2nd Highest   Hour Date/Time
        Count                                   Count
SYSA 258  1    Ø7 Feb Ø3 - Ø1:45  256       2    Ø7 Feb Ø3 - Ø2:ØØ
```

```
SYSB 162  2    21 Feb Ø3 - 12:3Ø  161      2    21 Feb Ø3 - 13:ØØ
CPC  388  1    Ø2 Feb Ø3 - 12:15  386      2    Ø2 Feb Ø3 - 12:ØØ
=================================================================
ACTIVE PRODUCT MAX CONTRIBUTORS
VWLC Product Name  VWLC Product ID    Highest   Date/Time    SYSA SYSB
z/OS V1       5694-AØ1      388      Ø2 Feb Ø3 - 12:15 233  155
DB2 UDB for OS/39Ø V7 5675-DB2       388      Ø2 Feb Ø3 - 12:15 233
155
CICS TS for OS/39Ø 5655-147      258      Ø2 Feb Ø3 - 12:15 258
IMS V7             5655-BØ1      162      21 Feb Ø3 - 12:3Ø      162

==================================================================
ACTIVE PRODUCT GRID SNAPSHOT

                             SYSA     SYSB
z/OS V1                 5694-AØ1    1ØØ%     1ØØ%
DB2 UDB for OS/39Ø V7   5675-DB2    1ØØ%     1ØØ%
CICS TS for OS/39Ø      5655-147     97%
IMS V7                  5655-BØ1             1ØØ%
===================================================================
Input Data Start Date Ø2 Feb 2ØØ3
Input Data End Date Ø2 Mar 2ØØ3

This report is prepared by the zSeries customer identified above
("Customer") or its authorized designee, and such Customer is solely
responsible for the contents and accuracy of this report. Specifically,
IBM makes no representations or warranties regarding the contents or
accuracy of this report. Any questions concerning this report should be
directed to the Customer.
```

Sending the SCRT report to IBM

When the SCRT report is produced, it must be sent to IBM.

The official submission process requires you to attach all your reports to an e-mail message and send it to the appropriate IBM e-mail address for your country.

The IBM e-mail address for each country can be found at http://www.ibm.com/zseries/swprice/scrt.

When a sub-capacity report is received, IBM sends a letter of acknowledgement to the report submitter, as indicated in the report. In some cases the letter will request additional information.

Systems Programmer
(France) © Xephon 2003

Parsing and validating MVS dataset names in REXX

Using the REXX language more amd more frequently over the past years, I have written many EXECs that make use of MVS dataset names as parameters. In the past, I have just used them without validating the dataset names, relying on MVS allocation to perform the validation and issue appropriate return codes. I recently decided to write a REXX function to perform dataset name parsing and validation, because I wanted to handle such validation as part of my own code rather than leaving it to MVS allocation. The result is the DSNOK REXX function, listed below.

The function is called simply by passing the dataset name to it as its only parameter, as follows:

```
dsn = "TEST.DATA.SET.NAME"
dsnrc = dsnok(dsn)
```

The function sets a return code, as documented at the beginning of the REXX function code, to indicate either success, or the cause of dataset name validation failure. The function handles dataset names in exactly the same way as the standard TSO parsing routine, IKJPARS. That is, if the dataset name passed is not enclosed in single quotes, the routine will fully qualify the dataset name by prefixing it with the current TSO prefix for the logged on TSO user. If the dataset name is enclosed in quotes, it will be treated as already fully qualified. In the example above, the dataset name is not considered fully qualified. In order to make it so, it would need to be coded as follows:

```
dsn = "'TEST.DATA.SET.NAME'"
```

The routine can also handle the inclusion of a PDS member name as part of the dataset name. It does not, however, verify that the dataset specified actually exists. Such code can be included in the routine, or added as separate code after receiving a zero return code from the function call itself.

After I wrote the REXX function, I started to think about the situation where I might have a large number of dataset names to validate. Since I do not have a REXX compiler available, I decided, as an academic exercise, to write an Assembler REXX function to perform the same dataset name parsing. I used the IKJPARS service mentioned earlier to perform the validation, rather than writing my own Assembler parsing code. Predictably, for repetitive validation calls, the Assembler function performs better than the REXX function. It is called in the same fashion as DSNOK, that is:

```
     a = asmpars(dsn)
```

and returns to the caller with the return code provided by IKJPARS, with the exception of some initial parameter validations return codes. If no parameter is provided to the function, a return code of 1 is returned. If the dataset name is over the maximum allowed, a return code of 2 is returned. Finally, a return code of 3 is returned if multiple arguments are supplied to the function, but the first argument is null.

DSNOK REXX SUBROUTINE

```
/* rexx comment *** start standard header
PARSEDSN Test use of imbedded rexx function DSNOK for dsn verification
                               rexx comment *** end   standard header */

dsnok:
/* This function set the following return codes:
 Ø = valid dataset name found
 1 = no dataset name supplied
 2 = unbalanced quotes
 3 = invalid dataset name length
 4 = invalid member specification
 5 = invalid dataset qualifier length
 6 = invalid dataset qualifier characters (start)
 7 = invalid dataset qualifier characters (rest)
 8 = invalid dataset last qualifier
 9 = invalid member name length
1Ø = invalid member name characters
2Ø = invalid dataset last characters
*/
parse arg dsn
dsnlen = length(dsn)
select                              /* check for paired single quotes  */
   when dsnlen = Ø then return 1             /* no dataset name given  */
   when substr(dsn,1,1) = "'" & ,           /* starts & ends with "'" */
        substr(dsn,dsnlen,1) = "'" then do
        dsn = strip(dsn,"B","'")            /* remove start/end "'"   */
        dsnlen = length(dsn)                /*  and adjust dsn length */
        end /* when */
   when substr(dsn,1,1) = "'" | ,           /* just starts or just    */
        substr(dsn,dsnlen,1) = "'" then ,   /*  ends with "'"         */
        return 2
   otherwise do                     /* othewise no quotes to process */
        dsnpref = sysvar(syspref)   /* check for a prefix            */
        if dsnpref ¬= "" then do
           dsn = dsnpref"."dsn      /* fully qualify the dsn as      */
           dsnlen = length(dsn)     /*    per TSO dsn parsing rules  */
           end /* if dsnpref */
        end /* otherwise */
end /* select */
delim = 'ØØ'x                     /* for string termination checking */
parse value dsn||delim with dsn "(" member ")" rest /*split dsn/member*/
dsn = strip(dsn,"T",delim)        /* strip delim at end of dsn       */
dsnlen = length(dsn)
if dsnlen > 44 | dsnlen < 1 then return 3   /* check dsn length      */
select
   when rest = delim & member ¬= "" then validmem = 1     /* (member) */
   when member ¬= "" then return 4          /* (member or (member)... */
   otherwise nop
end /* select */
dsntemp = dsn
charlist1 = "ABCDEFGHIJKLMNOPQRSTUVWXYZ@#$" /* valid 1st character    */
charlist2 = charlist1||'Ø123456789'         /* valid other characters */
do while dsntemp ¬= ""                      /* validate dsn qualifiers*/
   parse var dsntemp dsnqual "." dsntemp    /* split off a qualifier  */
   quallen = length(dsnqual)
   if quallen > 8 | quallen < 1 then return 5 /* check qualifier len  */
   if pos(left(dsnqual,1),charlist1) = Ø   ,  /* check qualifier chars*/
      then return 6
   if quallen > 1 then ,                      /* check qual rest chars*/
      if verify(substr(dsnqual,2,quallen-1),charlist2,"N") ¬= Ø ,
         then return 7
end /* do while */
if substr(dsn,dsnlen,1) = "." then return 2Ø  /* dsn ended with a '.' */
if member ¬= "" then do           /* check member name, if present   */
   quallen = length(member)
   if quallen > 8 | quallen < 1 then return 9 /* check member length  */
   if pos(left(member,1),charlist1) = Ø | ,   /* check member chars   */
      verify(substr(member,2,quallen-1),charlist2) ¬= Ø then return 1Ø
end /* if member */
return Ø
```

ASMPARS ASSEMBLER PROGRAM

```
*%A LOAD('TSOBBOR.ISPLLIB') AMODGEN MAC3('TSOBBOR.MACLIB')
ASMPARS  CSECT
         YREGS
ASMPARS  AMODE 31
ASMPARS  RMODE ANY
         SAVE  (14,12),,ASMPARS-&SYSDATE
         LR    R12,R15
         USING ASMPARS,R12
         ST    R13,SAVEAREA+4
         LA    R11,SAVEAREA
         ST    R11,8(,R13)
         LA    R13,SAVEAREA
* WORK WITH REXX PARAMETERS
         LR    R11,R1                      SAVE A(EFPL)
         USING EFPL,R11
         L     R7,EFPLEVAL                 LOAD A(A(EVAL BLOCK))
         L     R7,Ø(,R7)                   LOAD A(EVAL BLOCK)
         USING EVALBLOCK,R7
         L     R1Ø,EFPLARG                 LOAD A(1ST ARG)
         USING ARGTABLE_ENTRY,R1Ø
         CLC   ARGTABLE_ENTRY,HIGHV        CHECK IF AT END OF TABLE
         BE    NOARGS                      YES, SET NO ARGS RC
         L     R8,ARGTABLE_ARGSTRING_PTR   LOAD A(ARG DATA)
         L     R9,ARGTABLE_ARGSTRING_LENGTH LOAD LENGTH(ARG)
         C     R9,CBUFDLEN                 IS ARG > MAX LEN
         BH    BADARGL1                    YES, SET BADLEN RC
         LTR   R9,R9
         BZ    BADARGL2                    ARGS, BUT LEN(ARG1)=Ø
         BCTR  R9,RØ                       DECREMENT FOR EXECUTE
         EX    R9,MVCPARM                  MOVE ARG TO CBUFDSNM
* BUILD PARAMETERS FOR CALLING IKJPARS
         L     R1,CVTPTR                   LOAD A(CVT)
         L     R1,CVTTCBP-CVT(R1)          LOAD A(TCBWORDS)
         L     R1,12(R1)                   LOAD A(CURRENT ASCB)
         L     R2,ASCBASXB-ASCB(R1)        LOAD A(ASXB)
         L     R2,ASXBLWA-ASXB(R2)         LOAD A(LWA)
         LA    R4,LPPL                     LOAD A(LOCAL PPL)
         USING PPL,R4
         MVC   PPLECT,LWAPECT-LWA(R2)      STORE A(LWA)
         L     R2,LWAPSCB-LWA(R2)
         MVC   PPLUPT,PSCBUPT-PSCB(R2)     STORE A(UPT)
         LA    R1,LECB
         ST    R1,PPLECB                   STORE A(ECB)
         L     R1,IKJADCON
         ST    R1,PPLPCL                   STORE A(PCL)
         LA    R1,LANS
         ST    R1,PPLANS                   STORE A(ANS)
         LA    R1,CBUF
         ST    R1,PPLCBUF                 STORE A(CBUF)
         L     R1,PPLUPT                  LOAD A(UPT)
         MVC   UPTSWSSV,UPTSWS-UPT(R1)    SAVE UPT SWITCHES
         OI    UPTSWS-UPT(R1),UPTNPRM     SET TO DISALLOW PROMPTING
         L     R1,PPLECT                  LOAD A(UPT)
         MVC   ECTSWSSV,ECTSWS2-ECT(R1)   SAVE ECT SWITCHES
         OI    ECTSWS2-ECT(R1),ECTNOPUT   SET TO DISALLOW PUTLINES
         LA    R1,PPL                     LOAD A(PLIST)
* NOTE THAT IKJPARS DOES NOT CARE ABOUT THE CASE OF THE OPERAND, BUT
* DYNAMIC ALLOCATION DOES
         LINK  EP=IKJPARS                 INVOKE PARSER
         L     R1,PPLUPT
         MVC   UPTSWS-UPT(Ø,R1),UPTSWSSV  RESTORE SETTING
         L     R1,PPLECT
         MVC   ECTSWS2-ECT(Ø,R1),ECTSWSSV RESTORE SETTING
         CVD   R15,DBLWD
         UNPK  PARSERC(4),DBLWD+6(3)
         MVC   EVALBLOCK_EVLEN,=F'2'
         MVC   EVALBLOCK_EVDATA(2),PARSERC
         B     EXIT
NOARGS   MVC   EVALBLOCK_EVLEN,=F'2'
         MVC   EVALBLOCK_EVDATA(2),=C'Ø1'
         B     EXIT
BADARGL1 MVC   EVALBLOCK_EVLEN,=F'2'
         MVC   EVALBLOCK_EVDATA(2),=C'Ø2'
         B     EXIT
BADARGL2 MVC   EVALBLOCK_EVLEN,=F'2'
         MVC   EVALBLOCK_EVDATA(2),=C'Ø3'
         B     EXIT
EXITRC   MVC   EVALBLOCK_EVLEN,=F'2'
         MVC   EVALBLOCK_EVDATA(2),=C'ØØ'
EXIT     L     R13,4(,R13)
         RETURN (14,12),RC=Ø
SAVEAREA DS    18F
DBLWD    DC    D'Ø'
IKJADCON DC    V(PARSPCL)
LECB     DC    F'Ø'
LANS     DC    F'Ø'
LPPL     DC    XL(PPLLEN)'ØØ'
HIGHV    DC    XL8'FFFFFFFFFFFFFFFF'
         DS    ØF
CBUFDLEN DC    A(L'CBUFDSNM)
CBUF     DC    AL2(CBUFLEN,CBUFDSNM-CBUFCMD)
CBUFCMD  DC    C'DUMYCMND '
CBUFDSNM DC    CL56' '          TO ALLOW FOR 'FULLY.QUAL.DSNAME(MEMBER)'
CBUFLEN  EQU   *-CBUF
MVCPARM  MVC   CBUFDSNM(Ø),Ø(R8)
PARSERC  DC    CL4' '
```

```
UPTSWSSV DS    X
ECTSWSSV DS    X
         LTORG
PARSPCL  IKJPARM
DSNAME   IKJPOSIT DSNAME,USID,PROMPT='DATASET NAME'
         IKJENDP
         CVT   DSECT=YES
         IHAASCB
         IHAASXB
         IKJEFLWA
         IKJPPL
PPLLEN   EQU   *-PPL
         IKJECT
         IKJPSCB
         IKJUPT
         IRXEFPL
         IRXARGTB
         IRXEVALB
         END
```

Systems Programmer (USA) © Xephon 2003

Creative use of VSAM datasets

Small applications can be efficiently created by using VSAM datasets. Instead of a database, like DB2 or Adabas, you can use VSAM to simulate some of the facilities. In this article I will describe (in a simplified fashion) a real example of an application created with just a KSDS and an RRDS. The approach is simple – the RRDS holds the data, and the KSDS holds the logical indexes that point to the data location (the RRDS record number). I will detail this later. For now I will describe what the application does and how it came about.

Every day we create a few dozen reports (listings) that the end users must access. Those reports are created in batch during the night. In a traditional environment, those listings would reside in the spool, and someone would have to physically print them, either locally or centrally. However, this approach involves a waste of paper, printing resources, and time. This is because several end users located in many different places may have to access these listings, sometimes only to consult a few lines and then forget them. If they have to print it, they will end up wasting paper (and time waiting for it, if the listing is large). On the other hand, if we give the users spool access (which normally we do not), this means more TSO users in the system, with all its inconveniences in terms of resources.

So we came up with a different idea: since the end users are all CICS users, why not put the listings on CICS? The only thing needed for that is to write the listings to a VSAM file and create a CICS application to access them.

And so the scheme works as follows: the batch programs that create the listings write them not to spool but to a temporary file. The next step in the JCL is a special COBOL program (LSTWRITE) that reads the temporary file (the listing), line by line, and writes those lines to an RRDS, starting at the next free RRDS record. Then it writes the index to that listing (its logical name, date, and a few other things) to a KSDS. All these index fields are part of the KSDS key, and the data in that KSDS record (the non-KSDS key part) is the number of the first and the last RRDS record where the listing was written. This means that just a KSDS and an RRDS dataset can hold as many listings as you like, provided both files are appropriately dimensioned.

And how does the program that writes the listing to the RRDS know which is the next RRDS available record (or the last used)? Simple. The number of the last relative record used is written in the first eight bytes (in character format) in RRDS record number one, which is used only for that purpose. The program simply starts by reading it, so it knows where to start writing. Once it finishes writing the listing, it updates the first RRDS record with the new last used record number.

Of course, if nothing is done, this scheme will grow indefinitely, so a periodic clean-up of old listings and an RRDS/KSDS reorganization is needed, to avoid leaving unused holes in the RRDS file. The concept of an old (or deletable) listing is flexible, but it usually means that the listing is over two weeks old. This reorganization is done once a week, and it consists of copying all non-deletable listings from the old RRDS file to a new one and also creating a new KSDS with the new pointers. Once this is done, the old VSAM files are deleted and the new renamed to the standard names. Again, a simple COBOL program (LSTREORG) performs this task.

Besides containing the content of the listing (up to 133 bytes, including the control character column), each RRDS line also contains at its beginning the logical key of the listing. This approach ensures that, in the eventuality that something goes wrong, the RRDS file alone has all the information needed to rebuild the entire KSDS.

The following scheme illustrates the relationship between the KSDS and the RRDS files:

```
KSDS:
KSDS key / logical key (32 bytes)          RRDS pointers:
(listing date is at position 9)       Start rec.       End rec.

LISTAAAB2ØØ2Ø5Ø5ACETTØØØØ56ØØØ12      ØØØØØØØ2         ØØØØØØ49
LISTCDEF2ØØ2Ø611BCKDYØØØ722ØØ981      ØØØØØØ5Ø         ØØØØØ13Ø
LISTGHIJ2ØØ2Ø5Ø8CSTRRØØØØ15ØØØØ9      ØØØØØ131         ØØØØØ194
```

```
RRDS:
(rec num.)    Copy of logical key               Listing data (133 bytes)

(Rec 1)       ØØØØØ194   (last used record)
(Rec 2)       LISTAAAB2ØØ2Ø5Ø5ACETTØØØØ56ØØØ12   (Listing contents)
(Rec 3)       LISTAAAB2ØØ2Ø5Ø5ACETTØØØØ56ØØØ12   …
…
(Rec 5Ø)      LISTCDEF2ØØ2Ø5Ø5BCKDYØØØ722ØØ981   (Listing contents)
(Rec 51)      LISTCDEF2ØØ2Ø5Ø5BCKDYØØØ722ØØ981   …
…
(Rec 131)     LISTGHIJ2ØØ2Ø611CSTRRØØØØ15ØØØØ9   (Listing contents)
(Rec 132)     LISTGHIJ2ØØ2Ø611CSTRRØØØØ15ØØØØ9   …
…
(Rec 195)     (free)
```

Both KSDS and RRDS files are known to CICS and are available during the day. The CICS application through which the users access the listings consists of a set of COBOL programs. Without going into much detail, when the end user enters the application, he is presented with a screen where he can enter some key fields of the listings he is seeking, like date, listing code, department, and so on. Normally, the user enters just a part of these fields, like the date and perhaps the starting characters of the listing title, optionally followed by an asterisk. All these fields are part of the KSDS key. The program seeks which keys match the user criteria and presents them, one key per line, in a scrollable fashion. This way, the user can choose which listing he wants to consult. All he has to do is place the cursor in the left part of the line that corresponds to it, in a special input area, where he can type commands. He can view, print, or transfer it to his PC.

If he chooses to view the listing, he enters another screen where he can browse the listing at will, back and forth, left or right, top or bottom. This is all controlled by a viewing program.

If he chooses to print it, he can choose any printer known to CICS, and he can decide how many copies to print and whether he wants to print all or just a few pages. The printing program honours all the control characters that exist in the listing left-column. In display mode, this column is not shown.

The transfer to PC option copies the listings contents to a CICS temporary storage queue with a suitable name, and then the user can transfer it using the standard mechanisms of the 3270 emulator he is using.

The above description of the application is a rather simplified one, in order to give you the basic ideas behind it. There is one aspect I have not focused on yet: the security mechanisms. Each user can access only a specific set of listings, for example those concerning their department. Other users can access from several departments, and some can access all of them. The security mechanism is implemented with the help of a third

KSDS file, and the first program of the CICS application takes care of that control. Also, not all users are allowed to download to a PC.

However, the details are not relevant to this article, which is to illustrate how two VSAM datasets can be used to solve a specific problem.

The CICS application is the most complex part of this scheme, and it contains many specific details, and even some interaction with other CICS applications that transparently call it and fill in parameters for the user so he can work in an integrated environment. I will not present that application here, since it is not directly relevant.

The two COBOL programs needed to maintain it (the one that copies listings to the RRDS and the program that reorganizes it) are very simple, and are presented here for reference, as well as the necessary JCL.

In this example, the logical key (or KSDS key) is 32 bytes long, but it could have any other value, provided the programs and the VSAM files are changed accordingly. Also, and for reorganization purposes (deleting of old listings), the date is located at key position 9, in yyyymmdd format, but it could be somewhere else.

The RRDS record length must be enough to contain the logical key plus the 133 bytes of the listing (if the listing is smaller, the rest should be padded with spaces).

The reorg JCL contains the definition of the target files. The base files (the initial ones) should have a similar definition.

You may notice that the write program opens the two VSAM files for output, closes them, and opens them again for I/O. This way, it does not matter whether the files are empty (open output would be required) or not (open I/O required), they will always work fine, because once they are opened for output and closed, they are no longer considered ‘empty’. I do not know exactly why this is so, I just know it works. With this trick, the only side-effect that you get is a pair of IEC161I messages (when the files are opened for output and they already contain data) that you can simply ignore.

EXAMPLE JCL TO INSERT A LISTING

```
//*==================================================
//* THIS STEP WRITES A LISTING TO A TEMPORARY FILE
//*==================================================
//*
//STEPA    EXEC PGM=program.that.creates.a.listing
//SYSPRINT DD SYSOUT=*
//LISTING  DD DSN=&&TEMP1,DISP=(NEW,PASS),
//         LRECL=133,RECFM=F,UNIT=VIO,
//         SPACE=(TRK,(15,15))
//*
//*==================================================
//* THIS STEP INSERTS THE LISTING IN THE RRDS / KSDS
//* WITH THE LOGICAL KEY SUPPLIED IN THE LISTKEY CARD
//*==================================================
//*
//STEPA    EXEC PGM=LSTWRITE
//SYSPRINT DD SYSOUT=*
//LISTKEY  DD *
LISTAAAB2ØØ2Ø5Ø5ACETTØØØØ56ØØØ12
/*
//FILEKSDS DD DISP=SHR,DSN=VSAMLIST.LSTKSDS
//FILERRDS DD DISP=SHR,DSN=VSAMLIST.LSTRRDS
//LISTDATA DD DISP=(OLD,DELETE),DSN=&&TEMP1
/*
```

EXAMPLE JCL TO DELETE OLD LISTINGS AND REORGANIZE THE DATASET

```
//*=========================================================
//* THIS STEP DEFINES THE NEW KSDS AND RRDS
//*=========================================================
//*
//STEPA    EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
 DEFINE CL                        -
         (NAME(VSAMLIST.LSTRRDS1)  -
          RECSZ(165 165)           -
          FSPC(Ø Ø)                -
          TRK(15Ø 15Ø)             -
          NUMBERED)                -
      DATA(NAME(VSAMLIST.LSTRRDS1.DATA))
 DEFINE CL                          -
          (NAME(VSAMLIST.LSTKSDS1)  -
           RECSZ(48 48)             -
           FSPC(1Ø 1Ø)              -
           TRK(5 5)                 -
           KEYS(32 Ø))              -
      DATA(NAME(VSAMLIST.LSTKSDS1.DATA)) -
     INDEX(NAME(VSAMLIST.LSTKSDS1.INDEX))
/*
//*============================================================
//* THIS STEP RUNS THE REORG PROGRAM, DELETING ALL LISTINGS
//* WHOSE DATE IS OLDER THAN THE ONE IN LASTDATE CARD
//*============================================================
//*
//STEPB    EXEC PGM=LSTREORG,COND=(4,LT,STEPA)
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//KSDSOLD  DD DISP=SHR,DSN=VSAMLIST.LSTKSDS
//RRDSOLD  DD DISP=SHR,DSN=VSAMLIST.LSTRRDS
//KSDSNEW  DD DISP=SHR,DSN=VSAMLIST.LSTKSDS1
//RRDSNEW  DD DISP=SHR,DSN=VSAMLIST.LSTRRDS1
//LASTDATE DD *
2ØØ2Ø9Ø1
/*
//*============================================================
//* THIS STEP DELETES THE OLD FILES AND RENAMES THE NEW ONES
//*============================================================
//*
//STEPC    EXEC PGM=IDCAMS,COND=(4,LT,STEPB)
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
 DELETE  (VSAMLIST.LSTRRDS) CL PURGE
 DELETE  (VSAMLIST.LSTKSDS) CL PURGE
 ALTER    VSAMLIST.LSTRRDS1         -
 NEWNAME (VSAMLIST.LSTRRDS)
 ALTER    VSAMLIST.LSTRRDS1.DATA    -
 NEWNAME (VSAMLIST.LSTRRDS.DATA)
 ALTER    VSAMLIST.LSTKSDS1         -
 NEWNAME (VSAMLIST.LSTKSDS)
 ALTER    VSAMLIST.LSTKSDS1.DATA    -
 NEWNAME (VSAMLIST.LSTKSDS.DATA)
 ALTER    VSAMLIST.LSTKSDS1.INDEX   -
 NEWNAME (VSAMLIST.LSTKSDS.INDEX)
/*
```

LSTWRITE SOURCE CODE

```
       IDENTIFICATION DIVISION.
```

```
 PROGRAM-ID. LSTWRITE.
*==============================================================
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 INPUT-OUTPUT  SECTION.
 FILE-CONTROL.
     SELECT LISTKEY  ASSIGN TO LISTKEY
            FILE STATUS FSØ.
     SELECT LISTDATA ASSIGN TO LISTDATA
            FILE STATUS FS1.
     SELECT FILEKSDS ASSIGN TO FILEKSDS
            ORGANIZATION INDEXED
            ACCESS DYNAMIC
            RECORD KEY FILEKSDS-KEY
            FILE STATUS FS2.
     SELECT FILERRDS ASSIGN TO FILERRDS
            ORGANIZATION RELATIVE
            ACCESS DYNAMIC
            RELATIVE RRDSKEY
            FILE STATUS FS3.
*
*==============================================================
 DATA DIVISION.
 FILE SECTION.
*==============================================================
*
 FD  LISTKEY
     BLOCK CONTAINS Ø RECORDS
     RECORDING MODE IS F
     LABEL RECORD STANDARD.
 Ø1  LISTKEY-FD.
     Ø2 FILLER          PIC X(8Ø).
 FD  LISTDATA
     BLOCK CONTAINS Ø RECORDS
     RECORDING MODE IS F
     LABEL RECORD STANDARD.
 Ø1  LISTDATA-FD.
     Ø2 FILLER          PIC X(133).
 FD  FILEKSDS.
 Ø1  FILEKSDS-FD.
     Ø2 FILEKSDS-KEY    PIC X(32).
     Ø2 FILLER          PIC X(16).
 FD  FILERRDS.
 Ø1  FILERRDS-FD.
     Ø2 FILLER          PIC X(165).
*
*==============================================================
 WORKING-STORAGE SECTION.
*==============================================================
*
 77  FSØ          PIC 99       VALUE Ø.
 77  FS1          PIC 99       VALUE Ø.
 77  FS2          PIC 99       VALUE Ø.
 77  FS3          PIC 99       VALUE Ø.
 77  RRDSKEY      PIC 9(8)     VALUE Ø.
 77  RRDSKEY1     PIC 9(8)     VALUE Ø.
 77  RRDSKEY2     PIC 9(8)     VALUE Ø.
 77  LINECOUNT    PIC 9(8)     VALUE Ø.
*
 Ø1  LISTKEY-WS.
     Ø2   LISTKEY-KEY-WS      PIC X(32).
     Ø2   FILLER              PIC X(48).
*
 Ø1  LISTDATA-WS.
     Ø2   FILLER              PIC X(133).
*
 Ø1  FILEKSDS-WS.
     Ø2   FILEKSDS-KEY-WS     PIC X(32).
     Ø2   FILEKSDS-DATA1-WS   PIC 9(8).
     Ø2   FILEKSDS-DATA2-WS   PIC 9(8).
*
 Ø1  FILERRDS-WS.
     Ø2   FILERRDS-WS1.
       Ø4 FILERRDS-LAST       PIC 9(8).
       Ø4 FILLER              PIC X(157).
     Ø2   FILERRDS-WS2 REDEFINES FILERRDS-WS1.
       Ø4 FILERRDS-KEY-WS     PIC X(32).
       Ø4 FILERRDS-DATA-WS    PIC X(133).
*
*==============================================================
 PROCEDURE DIVISION.
*==============================================================
*
 OPEN-FILES.
*===========*
     OPEN INPUT  LISTKEY  LISTDATA
          OUTPUT FILERRDS FILEKSDS.
     CLOSE    FILERRDS  FILEKSDS.
     OPEN I-O FILERRDS  FILEKSDS.
     IF FSØ NOT ZERO
        DISPLAY 'ERROR OPENING LISTKEY ' FSØ
        MOVE 99 TO RETURN-CODE
        GO TO PROGRAM-EXIT
     END-IF.
     IF FS1 NOT ZERO
        DISPLAY 'ERROR OPENING LISTDATA ' FS1
        MOVE 99 TO RETURN-CODE
        GO TO PROGRAM-EXIT
```

```
          END-IF.
          IF FS2 NOT ZERO
             DISPLAY 'ERROR OPENING FILEKSDS ' FS2
             MOVE 99 TO RETURN-CODE
             GO TO PROGRAM-EXIT
          END-IF.
          IF FS3 NOT ZERO
             DISPLAY 'ERROR OPENING FILERRDS ' FS3
             MOVE 99 TO RETURN-CODE
             GO TO PROGRAM-EXIT
          END-IF.
          MOVE SPACES TO LISTDATA-WS.
     *
      READ-LISTKEY-CHECK-KSDS.
     *=========================*
          READ LISTKEY INTO LISTKEY-WS
          IF FSØ NOT ZERO
             DISPLAY 'ERROR READING LISTKEY ' FSØ
             MOVE 99 TO RETURN-CODE
             GO TO PROGRAM-EXIT
          END-IF.
          MOVE LISTKEY-KEY-WS TO FILERRDS-KEY-WS
                                 FILEKSDS-KEY-WS
                                 FILEKSDS-KEY
          READ FILEKSDS
          IF FS2 EQUAL ZERO
             DISPLAY 'KEY ALREADY EXISTS IN KSDS'
             MOVE 99 TO RETURN-CODE
             GO TO PROGRAM-EXIT
          END-IF.
     *
      READ-FILERRDS-FIRST-RECORD.
     *============================*
          MOVE 1 TO RRDSKEY
          READ FILERRDS INTO FILERRDS-WS
          IF FS3 NOT ZERO
             MOVE SPACES TO FILERRDS-WS
             MOVE 1 TO FILERRDS-LAST
             WRITE FILERRDS-FD FROM FILERRDS-WS
             IF FS3 NOT ZERO
                DISPLAY 'ERROR CREATING REC 1 IN RRDS ' FS3
                MOVE 99 TO RETURN-CODE
                GO TO PROGRAM-EXIT
             END-IF
          END-IF.
          MOVE FILERRDS-LAST TO RRDSKEY RRDSKEY1 RRDSKEY2
          ADD 1 TO RRDSKEY1 RRDSKEY2
          MOVE LISTKEY-KEY-WS TO FILERRDS-KEY-WS.
     *
```

```
 READLOOP.
*=========*
     READ LISTDATA INTO LISTDATA-WS
          AT END
             IF LINECOUNT GREATER Ø
                PERFORM WRITE-KSDS-UPDATE-RRDS-FIRST
             END-IF
             GO TO PROGRAM-EXIT.
     MOVE LISTDATA-WS TO FILERRDS-DATA-WS
     ADD 1 TO RRDSKEY LINECOUNT
     WRITE FILERRDS-FD FROM FILERRDS-WS
     IF FS3 NOT ZERO
        DISPLAY 'ERROR WRITING FILERRDS ' FS3
        MOVE 99 TO RETURN-CODE
        GO TO PROGRAM-EXIT
     END-IF.
     GO TO READLOOP.
*
 WRITE-KSDS-UPDATE-RRDS-FIRST.
*============================*
     MOVE RRDSKEY  TO RRDSKEY2
     MOVE RRDSKEY1 TO FILEKSDS-DATA1-WS
     MOVE RRDSKEY2 TO FILEKSDS-DATA2-WS
     WRITE FILEKSDS-FD FROM FILEKSDS-WS
     IF FS2 NOT ZERO
        DISPLAY 'ERROR WRITING FILEKSDS ' FS2
        MOVE 99 TO RETURN-CODE
        GO TO PROGRAM-EXIT
     END-IF.
*
     MOVE 1 TO RRDSKEY
     READ FILERRDS INTO FILERRDS-WS
     MOVE RRDSKEY2 TO FILERRDS-LAST
     REWRITE FILERRDS-FD FROM FILERRDS-WS
     IF FS3 NOT ZERO
        DISPLAY 'ERROR REWRITING REC 1 IN RRDS ' FS3
        MOVE 99 TO RETURN-CODE
        GO TO PROGRAM-EXIT
     END-IF.
*
 PROGRAM-EXIT.
*============*
     IF RETURN-CODE EQUAL Ø
        IF LINECOUNT EQUAL Ø
           DISPLAY 'EMPTY LISTING - NO LISTING CREATED'
        ELSE
           DISPLAY '*** LISTING CREATED ***'
        END-IF
     END-IF.
```

```
           CLOSE  LISTKEY LISTDATA FILEKSDS FILERRDS
           STOP RUN.
```

LSTREORG SOURCE CODE

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. LSTREORG.
      *=============================================================
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       INPUT-OUTPUT  SECTION.
       FILE-CONTROL.
           SELECT KSDSOLD ASSIGN TO KSDSOLD
                  ORGANIZATION INDEXED
                  ACCESS SEQUENTIAL
                  RECORD KEY KSDSOLD-KEY
                  FILE STATUS FSØ.
           SELECT KSDSNEW ASSIGN TO KSDSNEW
                  ORGANIZATION INDEXED
                  ACCESS DYNAMIC
                  RECORD KEY KSDSNEW-KEY
                  FILE STATUS FS1.
           SELECT RRDSOLD ASSIGN TO RRDSOLD
                  ORGANIZATION RELATIVE
                  ACCESS DYNAMIC
                  RELATIVE OLDKEY
                  FILE STATUS FS2.
           SELECT RRDSNEW ASSIGN TO RRDSNEW
                  ORGANIZATION RELATIVE
                  ACCESS DYNAMIC
                  RELATIVE NEWKEY
                  FILE STATUS FS3.
           SELECT LASTDATE ASSIGN TO LASTDATE
                  FILE STATUS FS4.
      *
      *=============================================================
       DATA DIVISION.
       FILE SECTION.
      *=============================================================
      *
       FD  KSDSOLD.
       Ø1  KSDSOLD-FD.
           Ø2 KSDSOLD-KEY    PIC X(32).
           Ø2 FILLER         PIC X(16).
       FD  KSDSNEW.
       Ø1  KSDSNEW-FD.
           Ø2 KSDSNEW-KEY    PIC X(32).
           Ø2 FILLER         PIC X(16).
```

```
 FD  RRDSOLD.
 Ø1  RRDSOLD-FD.
     Ø2 FILLER         PIC X(165).
 FD  RRDSNEW.
 Ø1  RRDSNEW-FD.
     Ø2 FILLER         PIC X(165).
 FD  LASTDATE
     BLOCK CONTAINS Ø RECORDS
     RECORDING MODE IS F
     LABEL RECORD STANDARD.
 Ø1  LASTDATE-FD.
     Ø2 FILLER         PIC X(8Ø).
*
*============================================================
 WORKING-STORAGE SECTION.
*============================================================
*
 77  FSØ          PIC 99       VALUE Ø.
 77  FS1          PIC 99       VALUE Ø.
 77  FS2          PIC 99       VALUE Ø.
 77  FS3          PIC 99       VALUE Ø.
 77  FS4          PIC 99       VALUE Ø.
 77  OLDKEY       PIC 9(8)     VALUE Ø.
 77  NEWKEY       PIC 9(8)     VALUE Ø.
 77  NEWKEY1      PIC 9(8)     VALUE Ø.
 77  INCOUNT      PIC 9(8)     VALUE Ø.
 77  OUTCOUNT     PIC 9(8)     VALUE Ø.
*
 Ø1  KSDSOLD-WS.
     Ø2   KSDSOLD-KEY-WS.
       Ø4 FILLER             PIC X(8).
       Ø4 LISTDATE           PIC 9(8).
       Ø4 FILLER             PIC X(16).
     Ø2   KSDSOLD-DATA1-WS   PIC 9(8).
     Ø2   KSDSOLD-DATA2-WS   PIC 9(8).
*
 Ø1  KSDSNEW-WS.
     Ø2   KSDSNEW-KEY-WS     PIC X(32).
     Ø2   KSDSNEW-DATA1-WS   PIC 9(8).
     Ø2   KSDSNEW-DATA2-WS   PIC 9(8).
*
 Ø1  RRDSOLD-WS.
     Ø2   RRDSOLD-WS1.
       Ø4 RRDSOLD-LAST       PIC 9(8).
       Ø4 FILLER             PIC X(157).
     Ø2   RRDSOLD-WS2 REDEFINES RRDSOLD-WS1.
       Ø4 RRDSOLD-KEY-WS     PIC X(32).
       Ø4 FILERRDS-DATA-WS   PIC X(133).
*
```

```
 Ø1  RRDSNEW-WS.
     Ø2   RRDSNEW-WS1.
       Ø4 RRDSNEW-LAST       PIC 9(8).
       Ø4 FILLER             PIC X(157).
     Ø2   RRDSNEW-WS2 REDEFINES RRDSNEW-WS1.
       Ø4 RRDSNEW-KEY-WS     PIC X(32).
       Ø4 FILERRDS-DATA-WS   PIC X(133).
*
 Ø1  LASTDATE-WS.
     Ø2   LASTDT             PIC 9(8).
     Ø2   FILLER             PIC X(72).
*
*=============================================================
 PROCEDURE DIVISION.
*=============================================================
*
 OPEN-FILES.
*===========*
     OPEN INPUT  KSDSOLD RRDSOLD LASTDATE
          OUTPUT KSDSNEW RRDSNEW.
     IF FSØ NOT ZERO
        DISPLAY 'ERROR OPENING KSDSOLD ' FSØ
        MOVE 99 TO RETURN-CODE
        GO TO PROGRAM-EXIT
     END-IF.
     IF FS1 NOT ZERO
        DISPLAY 'ERROR OPENING KSDSNEW' FS1
        MOVE 99 TO RETURN-CODE
        GO TO PROGRAM-EXIT
     END-IF.
     IF FS2 NOT ZERO
        DISPLAY 'ERROR OPENING RRDSOLD' FS2
        MOVE 99 TO RETURN-CODE
        GO TO PROGRAM-EXIT
     END-IF.
     IF FS3 NOT ZERO
        DISPLAY 'ERROR OPENING RRDSNEW' FS3
        MOVE 99 TO RETURN-CODE
        GO TO PROGRAM-EXIT
     END-IF.
     IF FS4 NOT ZERO
        DISPLAY 'ERROR OPENING LASTDATE' FS4
        MOVE 99 TO RETURN-CODE
        GO TO PROGRAM-EXIT
     END-IF.
*
 READ-LASTDATE.
*==============*
     READ LASTDATE INTO LASTDATE-WS
     IF FS4 NOT ZERO
        DISPLAY 'ERROR READING LASTDATE ' FS4
        MOVE 99 TO RETURN-CODE
        GO TO PROGRAM-EXIT
     END-IF.
     MOVE 1 TO NEWKEY
     MOVE LOW-VALUES TO KSDSNEW-WS RRDSNEW-WS.
*
 READ-KSDSOLD-LOOP.
*==================*
     READ KSDSOLD INTO KSDSOLD-WS
          AT END
                 IF OUTCOUNT GREATER Ø
                    PERFORM WRITE-RRDS-FIRST
                 END-IF
                 GO TO PROGRAM-EXIT.
     ADD 1 TO INCOUNT
     IF LISTDATE < LASTDT
        GO TO READ-KSDSOLD-LOOP
     END-IF.
*
     MOVE KSDSOLD-DATA1-WS TO OLDKEY
     MOVE NEWKEY TO NEWKEY1
     ADD 1 TO NEWKEY1
     PERFORM READ-RRDSOLD-LOOP
             VARYING OLDKEY FROM KSDSOLD-DATA1-WS BY 1
             UNTIL   OLDKEY > KSDSOLD-DATA2-WS
     PERFORM WRITE-KSDSNEW
     GO TO READ-KSDSOLD-LOOP.
*
 READ-RRDSOLD-LOOP.
*==================*
     READ RRDSOLD INTO RRDSOLD-WS.
     IF FS2 NOT ZERO
        DISPLAY 'ERROR READING RRDSOLD ' FS2 ' ' OLDKEY
        MOVE 99 TO RETURN-CODE
        GO TO PROGRAM-EXIT
     END-IF.

     MOVE RRDSOLD-WS TO RRDSNEW-WS
     ADD 1 TO NEWKEY
     WRITE RRDSNEW-FD FROM RRDSNEW-WS
     IF FS3 NOT ZERO
        DISPLAY 'ERROR WRITING RRDSNEW ' FS3
        MOVE 99 TO RETURN-CODE
        GO TO PROGRAM-EXIT
     END-IF.
*
 WRITE-KSDSNEW.
```

```
*==============*
     MOVE KSDSOLD-WS TO KSDSNEW-WS
     MOVE NEWKEY1 TO KSDSNEW-DATA1-WS
     MOVE NEWKEY  TO KSDSNEW-DATA2-WS
     WRITE KSDSNEW-FD FROM KSDSNEW-WS
     IF FS2 NOT ZERO
        DISPLAY 'ERROR WRITING KSDSNEW' FS2 ' ' KSDSNEW-WS
        MOVE 99 TO RETURN-CODE
        GO TO PROGRAM-EXIT
     END-IF.
     ADD 1 TO OUTCOUNT.
*
 WRITE-RRDS-FIRST.
*=================*
     MOVE SPACES TO RRDSNEW-WS
     MOVE KSDSNEW-DATA2-WS TO RRDSNEW-LAST
     MOVE 1 TO NEWKEY
     WRITE RRDSNEW-FD FROM RRDSNEW-WS
     IF FS3 NOT ZERO
        DISPLAY 'ERROR WRITING RRDSNEW REC 1 ' FS3
        MOVE 99 TO RETURN-CODE
        GO TO PROGRAM-EXIT
     END-IF.
*
 PROGRAM-EXIT.
*=============*
     DISPLAY '*** REORGANIZATION CONCLUDED ***'.
     DISPLAY 'LISTINGS READ......: ' INCOUNT.
     DISPLAY 'LISTINGS WRITTEN...: ' OUTCOUNT.
     CLOSE KSDSOLD RRDSOLD KSDSNEW RRDSNEW LASTDATE
     STOP RUN.
```

Systems Programmer
(Portugal)

© Xephon 2003

BPXMTEXT utility

As non-Unix users, we are constantly challenged by the new USS world and make lots of stupid errors! Deciphering the resultant error messages is not really intuitive. No message numbers, eight digit error codes, etc.

But IBM has delivered a ‘goody’ to help us! SYS1.SBPXEXEC(BPXMTEXT) contains an EXEC that displays short explanation messages.

For example, if you try to dismount SYS1.ROOT using ISHELL, you get the following message, which is not very clear:

```
                                Work with Mounted File Systems
  .-----------------------------------------------.
S |           Unmount the File System             | es.
U |                                               | t or quiesce
  | CAUTION:                                      | us       Row 1 of 11
_ | The file system is about to be unmounted.     | lable
_ | File system name:                             | lable
_ | SYS1.ROOT                                     | lable
_ |                                               | lable
_ | Unmount option:                               | lable
_ | __  1.  Normal                                | lable
_ |     2.  Drain                                 | lable
_ |     3.  Immediate                             | lable
_ |     4.  Force                                 | lable
_ |                                               | lable
u | Drain wait time . . . . . 6Ø    seconds       | lable
  |                                               |
  |                                               |
  |                                               |
  .-------------------------------------------------------------------.
  | Errno=72x The resource is busy;
                              Reason=Ø588ØØAA The file system has file |
  | systems mounted on it.  Press Enter to continue.                   |
  '-------------------------------------------------------------------'
```

Ouch! What is the meaning of the reason code 058800AA?

With BPXMTEXT, it is easy to get the answer. You only have to enter the following command on the ISPF command line:

```
BPXMTEXT Ø588ØØAA
```

And you get the answer!

```
MT       BPXMTEXT msg: Ø588ØØAA
BPXFSUMT Ø6/Ø5/Ø2
JRFsParentFs: The file system has file systems mounted on it

Action: An unmount request can be honoured only if there are no file
systems mounted anywhere on the requested file system.
Use the D OMVS,FILE command
from the system console to find out which file systems are mounted on
the requested file system.  Unmount them before retrying this request.
***
```

Often this is enough information to determine the real error; if not, well there is always the ‘friendly’ manual!

Systems Programmer
(France) © Xephon 2003

MVS news

BMC has launched its System Explorer for z/OS, a GUI for the most commonly-used interactive functions of z/OS, and the first product in its new System Advisor series. The idea is to complement the company's system management applications.

With its interface similar to Microsoft Windows Explorer, it's said to simplify the most commonly-used functions for managing z/OS, including file management and job management, and provides a topological view of system resources.

System Explorer for z/OS is part of the Configuration Advisor line, which adds configuration management to the company's mainframe management capabilities.

For further information contact:
BMC, 2101 CityWest Blvd, Houston, TX 77042, USA.
Tel: (713) 918 8800.
URL: http://www.bmc.com/products/proddocview/0,2832,19052_19429_7418253_8998,00.html.

* * *

BMC has announced Mainview AutoOPERATOR for OS/390, which helps increase availability through automation and rules-based operations, and support IMS, CICS, WebSphere MQ, and TapeSHARE. Included with AutoOPERATOR for OS/390 Version 6.3.01 is new Total Object Manager (TOM), enabling IT administrators to use object management as an umbrella for the management and automation of IT resources that support certain business functions.

It lets users control IT assets across the sysplex better and creates the foundation for managing divergent objects with complex inter-dependencies, such as Unix System Services (USS) processes, MQ queues, and SAP applications.

BMC has also re-packaged its Mainview Storage Resource Manager (SRM) suite for managing performance and space consumption for various storage resources and processes. Its eight products are now available in three packages: Mainview SRM Reporting, Allocation, and Automation.

In addition, there are new features to expand OS/390 space management functions and reduce the incidence of space-related processing problems and new system-level functions that intercept abend conditions or standards violations, to provide services without any JCL changes.

For further information contact:
BMC, 2101 CityWest Blvd, Houston, TX 77042, USA.
Tel: (713) 918 8800.
URL: http://www.bmc.com/products/proddocview/0,2832,19052_19429_28229_8571,00.html.

* * *

IBM has announced z/OS V1.4 with enhancements that extend dynamic, flexible partitioning and resource management, availability, scalability, clustering, and Quality of Service (QoS). These enhancements include support for the zSeries z990, improved operator messaging architecture of z/OS, and concurrent access to VSAM datasets for both batch and CICS online transactions.

For further information contact your local IBM representative.
URL: http://www.ibm.com/servers/eserver/zseries.