155 VM update

July 1999

In this issue

VM Update

Published by
Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38030
From USA: 01144 1635 38030
E-mail: info@xephon.com

North American office
Xephon/QNA
1301 West Highway 407, Suite 201-405
Lewisville, TX 75077-2150
USA
Telephone: 940 455 7050

Editorial panel
Articles published in *VM Update* are reviewed by our panel of experts. Members of the panel include John Illingworth (UK), Reinhard Meyer (Germany), Philippe Taymans (Belgium), Romney White (USA), Martin Wicks (UK), and Jim Vincent (USA).

Subscriptions and back-issues
A year's subscription to *VM Update*, comprising twelve monthly issues, costs £180.00 in the UK; $275.00 in the USA and Canada; £186.00 in Europe; £192.00 in Australasia and Japan; and £190.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 1990 issue, are available separately to subscribers for £16.00 ($23.00) each including postage.

Editor
Robert Burgess

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

VM Update on-line
Code from *VM Update* can be downloaded from our Web site at http://www.xephon.com; you will need the user-id shown on your address label.

Contributions
Articles published in *VM Update* are paid for at the rate of £170 ($250) per 1000 words for original material. To find out more about contributing an article, without any obligation, please contact us at any of the addresses above and we will send you a copy of our *Notes for Contributors*.

Printed in England.

Monitoring executing programs

While working on Year 2000 problems over the last few years, I have found that some users have developed various cloned versions of production programs, and that these cloned versions reside on their mini-disks instead of our production mini-disk. Even worse, the users are often unaware of these local copies, because many were made some time ago and have been forgotten. Although we are going to ensure that programs on production disks are Year 2000-compliant, we anticipate problems with these 'undocumented' local copies.

While it would be possible simply to scan all users' mini-disks, a better approach is to establish a monitor of the programs being executed. In this way, we will avoid working on programs that are never used – this consideration applies to the programs on production disks as well.

The program CMGVAD performs just such monitoring. It must be activated to work, for example, from PROFILE EXEC. When activated, it NUCXLOADs itself and establishes system exit REXXEXIT. From that point, every time REXX is starting or terminating an application, it passes the control to CMGVAD with enabled EXECCOMM interface. CMGVAD retrieves PARSE SOURCE and ARG information from the EXEC being started. This information is sent via SMSG to a server virtual machine ('controller').

On the controller, there is another EXEC (CMGDOG) constantly running. CMGDOG accepts the messages and writes a log file. The information it has allows further reporting, such as the sample implemented statistics – which EXEC was executed and for how long.

CMGDOG writes the files with CSL routines. The reason for this complexity is that this method allows another user to see the current contents of the log file, not requiring CMGDOG to close it after each record.

CMGDOG

```
/**/
parse source . env fn ft fm fa address .
if address = 'CMS' then do
```

```
   'CP SET SMSG IUCV'
   'PIPE COMMAND EXECDROP CMGDOG REXX'
   'EXECLOAD' fn ft fm '= REXX'
   'PIPE STARMSG | REXX' fn fm    /* fm is a parm: output filemode */
   'EXECDROP' fn 'REXX'
    exit
   end

arg fm +1

fid = fn 'LOG' fm
p2 = 'WRITE NORECOVER NOCACHE V'
fn_ft = fn 'LOG'
call csl 'DMSOPEN RC RRC FID' length(fid) 'P2' length(p2) 'TOKENLOG'
   if rc<>0  & rrc<>440 30  then exit rc
fid1=fid
fid = fn 'STAT' fm
call csl 'DMSOPEN RC RRC FID' length(fid) 'P2' length(p2) 'TOKENSTAT'
   if rc<>0  & rrc<>440 30  then exit rc

say ,
 'Monitoring... results are in' fid1 'and' fid'. To finish, enter HMSG'

Do forever
 'READTO V'
    if rc=12 then leave
 parse var v 1 class +8 vmid +8 flag +1 'CMS' caller fn ft fm ,
             calledAs address x'0 0 ' parms
 out = left(date(),12) left(time(),8) left(vmid,8)  ,
         left(fn,8) left(ft,8) left(fm,2) left(calledAs,8) parms
 if flag = 'I' then do           /* init */
  stat.vmid.fn.time = time('S')
   stat.vmid.fn.date = date('B')
   Buf = 'S' out
   bufL = length(buf)
   call csl 'DMSWRITE RC RRC TOKENLOG 1 BUFL BUF BUFL 0  WU 0  FORCE 5'
    if rc>4 then signal error
 end

 else do                                    /* termination */
   if symbol('STAT.'vmid'.'fn'.TIME') = 'VAR'
   then do                                  /* calc elapsed time */
      elapsed = (date('B') - stat.vmid.fn.date)*24*360 0  + ,
                time('S') -  stat.vmid.fn.time
      eHours  = elapsed%360 0
      eMins   = (elapsed//360 0 ) % 60
      eSec    = (elapsed//60 )
      buf = 'E',
        left(subword(out,1,4),22)                                ,
       right(eHours,2,'0 ')':'right(emins,2,'0')':'right(esec,2,'0 ') ,
             '  '                                                ,
             subword(out,5)
```

```
      bufl = length(buf)
      call csl ,
       'DMSWRITE RC RRC TOKENSTAT 1 BUFL BUF BUFL Ø  WU Ø  FORCE 5'
       if rc>4 then signal error
      drop stat.vmid.fn.time stat.vmid.fn.date
    end
    buf = 'T' out; bufl = length(buf)
    call csl 'DMSWRITE RC RRC TOKENLOG 1 BUFL BUF BUFL Ø  WU Ø FORCE 5'
       if rc>4 then signal error
 end
 call csl 'DMSCOMM RC RRC'
end /* do forever */
call csl 'DMSCLOSE RC RRC TOKENSTAT COMMIT 6'
call csl 'DMSCLOSE RC RRC TOKENLOG  COMMIT 6'
```

CMGVAD

```
CMGVAD   CSECT   *
         SAVE    (14,12)
         USING   CMGVAD,R12
         LR      R12,R15
         ST      R13,SAVE+4
         LA      R13,SAVE
*                                      NUCXLOAD ME
         NUCEXT  CLR,NAME=MYNAME
         LA      R1,CMDNUCXLOAD
         MVC     CMDNUCXLOAD+8,MYNAME   MY NAME
         CMSCALL ERROR=ERR
*                                      FIGURE THE ENTRY ADDRESS
         NUCEXT  QUERY,NAME=MYNAME,ERROR=ERR  MY OWN NAME
         LR      R11,R1                 RETURNED SCBLOCK IN (R1)
         USING   SCBLOCK,R11
         L       R2,SCBENTR             NUCXLOAD ADDRESS
         LA      R2,(LØ -CMGVAD)(R2)    MAKE R2 -> ACTIVE ENTRY
*                                       DECLARE REXX EXITS
         REXEXIT SET,NAME='CMGVADI',ENTRY=(2),INIT=YES,ERROR=ERR,      -
                 SYSTEM=YES
         REXEXIT SET,NAME='CMGVADT',ENTRY=(2),TERM=YES,ERROR=ERR,      -
                 SYSTEM=YES
         B       EXIØ                   RETURN TO CMS
*
MYNAME   DC      CL8'CMGVAD'
CMDNUCXLOAD DC   CL8'NUCXLOAD',CL8' ',CL8'(',CL8'SYSTEM',8X'FF'
*-------------------------------------------------------------------
*      THE FOLLOWING PART TAKES CONTROL FROM REXX WHEN LAUNCHING
*      OR TERMINATING REXX APPLICATION.
*   -  ESTABLISH EXECCOMM INTERFACE
*   -  FIGURE THE NAME AND PARMS OF THE EXEC BEING STARTED
*   -  SMSG THAT INFO TO "CONTROLLER" VMID
LØ       EQU     *                     ACTIVE ENTRY
```

```
         SAVE    (14,12)
         USING   LØ ,12
         LR      R12,R15
         ST      R13,SAVE+4
         LA      R13,SAVE
*   EXTRACT ARG AND PARSE SOURCE INFO FROM THE EXEC BEING INITED

*                                    FILL SHVBLOCKS:SOURCE AND ARGS
         LR      R9,R1               PARMLIST ADDR -> R9
         LA      R1Ø ,SHV1           1ST
         USING   SHVBLOCK,R1Ø
         XC      SHV1,SHV1           CLEAR FIELDS
         MVC     SHVNEXT,=A(SHV2)    NEXT BLOCK ADDR
         OI      SHVCODE,SHVPRIV     FUNCTION: FETCH PRIV INFO
         MVC     SHVVALA,=A(PARM)
         MVC     SHVBUFL,=A(L'PARM)
         LA      R2,=C'ARG'
         ST      R2,SHVNAMA          PARSE ARG
         MVC     SHVNAML,=F'3'       (LENGTH OR 'ARG')
         LA      R1Ø ,SHV2           2ND
         XC      SHV2,SHV2           CLEAR FIELDS
         OI      SHVCODE,SHVPRIV     FUNCTION: FETCH PRIV INFO
         MVC     SHVVALA,=A(SOURCE)  OUTPUT BUFFER
         MVC     SHVBUFL,=A(L'SOURCE) BUFFER LENGTH
         LA      R2,=C'SOURCE'
         ST      R2,SHVNAMA          PARSE SOURCE
         MVC     SHVNAML,=F'6'       (LENGTH OF 'SOURCE')
         CMSCALL PLIST=PL,EPLIST=EPL, ISSUE EXECCOMM REQUEST         -
                 CALLTYP=SUBCOM,ERROR=ERR
         USING   PARMLIST,R9         INIT OR TERM? PUT IN MESSAGE
         CLC     EXITCODE,=X'Ø Ø Ø 9' IF EXITCODE<>'INIT' (Ø 9)
         BNE     L2
         MVI     INITORTERM,C'I'     THEN PUT CODE 'I'
         B       L3
L2       MVI     INITORTERM,C'T'     ELSE PUT CODE 'T'
L3       EQU     *
*                                    SMSG TO THE CONTROLLER
         LA      R1Ø ,SHV2           2ND SHVBLOCK: PARSE SOURCE
         LA      R4,L'SMSG           COMMAND LENGTH
         LA      R4,1(R4)            +1 FOR I/T (ABOVE)
         LA      R2,SMSGTEXT
         L       R3,SHVVALL          ARG LENGTH
         BCTR    R3,Ø                -1 FOR EX
         L       R5,SHVVALA          VALUE
         EX      R3,MVC1             MVC SMSGTEXT,SHVVAL
         A       R2,SHVVALL          OFFSET BUFFER LOC
         A       R4,SHVVALL          OUTPUT LENGTH
         LA      R1Ø ,SHV1           1ST SHVBLOCK: ARG
         L       R3,SHVVALL          SOURCE LENGTH
         LTR     R3,R3               NULL ARG?
         BZ      L1                  YES, NOTHING TO INSERT
```

```
*                                       NO, INSERT ARG IN SMSG
         MVI     Ø (R2),X'Ø Ø '         SEPARATE SOURCE/ARG W/X'Ø Ø '
         LA      R2,1(R2)               ADVANCE OUTPUT OFFSET
         LA      R4,1(R4)               LENGTH
         BCTR    R3,Ø                   -1 FOR EX
         L       R5,SHVVALA             VALUE ADDR
         EX      R3,MVC1                MVC SMSGTEXT,SHVVAL
         A       R4,SHVVALL             R4:=LEN(ARG)+LEN(SOURCE)+1
L1       EQU     *
         LA      R3,SMSG                SMSG COMMAND TEXT
         LR      R5,R4                  LENGTH
         O       R5,=X'4Ø Ø Ø Ø Ø Ø '   RETURN RESPONSE IN BUFFER
         LA      R4,RESPBUF             ADDR...
         LA      R6,L'RESPBUF           LENGTH...
         DIAG    R3,R5,X'Ø 8'           TRANSMIT INFO TO LOGGER
         B       EXIØ                   EXIT
         DS      Ø F                    FOR SPEED
MVC1     MVC     Ø (Ø ,R2),Ø (R5)
         B       EXIØ
ERR      EQU     *                      SMSG ERROR TO CONTROLLER
         BALR    R12,Ø
         USING   *,R12
         LA      R3,ERRMSG
         LA      R5,L'ERRMSG
         O       R5,=X'4Ø Ø Ø Ø Ø Ø '   RETURN RESPONSE IN BUFFER
         LA      R4,RESPBUF             ADDR...
         LA      R6,L'RESPBUF           LENGTH...
         DIAG    R3,R5,X'Ø 8'
         B       EXIØ
EXIØ     EQU     *
         BALR    R12,Ø
         USING   *,R12
         L       R13,SAVE+4
         XR      R15,R15                RC:=Ø
         RETURN  (14,12),RC=(15)
         DS      Ø D
SAVE     DS      18F
PL       DC      CL8'EXECCOMM'
EPL      DC      A(PL),A(Ø ),A(Ø ),A(SHV1)
SHV1     DS      CL32
SHV2     DS      CL32
SOURCE   DS      CL8Ø
PARM     DS      CL12Ø
SMSG     DC      C'SMSG CONTRLLR'       HARDCODED CONTROLLER'S NAME
INITORTERM DS    CL1
SMSGTEXT DS      CL2Ø Ø
ERRMSG   DC      C'SMSG CONTRLLR ERROR IN CMGVAD'
RESPBUF  DS      CL8Ø
PARMLIST DSECT
EXITNAME DS      CL8
EXITCODE DS      H
```

```
EXITSF   DS      CL2
EXITUW   DS      F

         EPLIST
         SHVBLOCK
         SCBLOCK
         REGEQU
         END
```

Vadim Rapp
Vadim Rapp Ltd (USA)

VM/ESA data-in-memory techniques

VM/ESA provides so many different techniques to put data in memory and/or to share storage that some people get a bit lost – for example, you may think that a VM dataspace can replace a saved segment.

All data-in-memory techniques are meant to boost the system's performance by reducing or eliminating I/Os or reducing real storage consumption via sharing.

TERMINOLOGY

Before we discuss the pros and cons of the different data in memory techniques, we will review some VM terminology:

- Module – a module is a CMS file containing an executable program. A module resides on a mini-disk or SFS directory and has to be loaded into storage before execution.
- Nucleus resident – most CMS commands are not modules, but are nucleus resident programs. That is, their coding is included in the CMS nucleus and, as the nucleus resides in a saved segment, their code does not need to be read from disk at each invocation.
- Nucleus extension – a module can be made resident in storage via the NUCXLOAD command. This means that it is loaded from disk to storage only once and can then repeatedly be invoked from there.

- Link-edit – the process that 'glues' together different parts of a program (such as subroutines). In CMS, those parts have a filetype of TEXT or are members of a TXTLIB. The process replaces the names of called subroutines with the address at which they are loaded. LOAD is the native CMS linkage editor and it creates MODULES. LKED is a CMS command that calls the MVS linkage editor and creates executables in a LOADLIB.
- Relocatable – a program is said to be relocatable if it can be executed at another address than the one at which it has been link-edited. The linkage editor can save the list of subroutines in the module, permitting the loader to adapt their addresses to the storage locations where they are loaded, just before execution. This is required for a program to be NUCXLOADed. A CMS program can be made relocatable by using the RLDSAVE option on the LOAD command.
- Reusable – a program is reusable when it can be re-executed without reload from disk. In practice, this means the program may have to re-initialize any area it might have changed during a previous execution. It is obvious that the programmer has to take care with this. This is also required for a program to be NUCXLOADed.
- Re-entrant – for a program to be re-entrant it must not modify itself, including data areas that are embedded within the program. To get a re-entrant program, the programmer has to use specific techniques. Because saved segments are read-only, re-entrancy is, of course, a requirement for a program to be included in a saved segment (otherwise modifications applied by one virtual machine would also influence the program's behaviour in other virtual machines using the same segment).
- Address space – is the addressable storage area where programs are executed and data resides. When a virtual machine is created, CP immediately creates this space. The size of this space is equal to the so-called VM size, defined in the CP directory, and can be altered by a CP DEFine STORage command. An address space is divided into segments of 1MB each in the XA architecture; segments were 64KB in 370 architecture.

SEGMENT TABLES, PAGE TABLES, AND PMB

Segment and page tables are built by CP and used by the hardware to describe the virtual storage virtual machines. A segment table entry points to a page table. A page table has 256 elements, and each entry describes the state of a virtual storage page.

With the page table entries, the hardware can find whether a page is in real storage, and where it is located. In VM/ESA, a page table is placed at the start of a Page Management Block (PMB). The PMB also includes information that allows you to find pages located on DASD (eg in the paging areas).

DATA SPACES

Data spaces are similar to address spaces, but contain only data. Programs cannot be executed directly from them. Data spaces can be shared among users and are defined by the operating system at the request of a program. Two types exist:

- ESA/370 data spaces are shareable only by operating systems running in paging mode (ie CP, MVS, and VSE). They can exist on any ESA-capable processor.
- ESA/390 VM data spaces are exclusive to VM and can be shared between operating systems running with Dynamic Address Translation OFF (ie CMS). These require an ES/9000 machine.

Both address spaces and data spaces are virtual storage and thus are pageable.

SAVED SEGMENT

A saved segment (often called a shared segment) is an area of an address space that can be shared among different virtual machines. Saved segments can contain programs and/or data. Note that a saved segment itself is non-relocatable in that it will always be loaded at the same virtual address where it was generated.

Saved segments can be loaded by Diagnose 64, or, better, by CMS' SEGMENT command or macro. When a segment is loaded, CP changes the segment table of the virtual machine to make one or more

entries point to the page tables of the saved segment. The pages of the saved segment are not directly paged in – this will only happen when users try to reference pages of the saved segment. When segment table entries for different virtual machines point to the same page tables, storage is shared. The information in the PMB will also guide CP's paging routines to page-in the pages from the spool (where the code of saved segments resides).

LSEG

A logical saved segment (LSEG) is a CMS concept that eases the inclusion of MODULEs, EXECs, etc into a saved segment. A classical segment is one big piece of coding, whereas an LSEG is a kind of library. When an LSEG gets loaded, all its objects become 'known' to the virtual machine. That is, MODULEs are considered NUCXLOADed, EXECs become EXECLOADed, etc. From then on, the fact that the elements are in a saved segment is transparent and the saved segment makes the code of the objects shareable among virtual machines.

An LSEG resides in a PSEG (physical segment). When loading an LSEG, CMS requests CP to get the appropriate PSEG. To CP, a PSEG is an ordinary saved segment. The SYSTEM SEGID file on the S disk defines the relationship between LSEGs and PSEGs. LSEGs are created with the SEGGEN command (VMFBLD can call SEGGEN as well).

FST

Each CMS-formatted mini-disk has a directory (list of the files and their attributes). When a CMS mini-disk (or an SFS directory) is accessed, this directory is copied from disk into the user's address space, where it is called a File Status Table (FST).

CU CACHEING

DASD control units can also keep data in their cacheing storage. When the data to be read is available from the cache, it is sent to the CPU roughly 10 times faster than when read from DASD. We won't

discuss this technique any further because it only speeds up I/O and has little to do with storage sharing. The Redbook *VM/ESA Storage Management with Tuning Guidelines* (GG24-3944) contains useful information in this area and is recommended reading.

MINI-DISK CACHEING

Mini-disk cacheing (MDC) is a CP service to avoid disk I/O. When a virtual machine reads a block from disk, CP saves a copy in real storage. From then on, any user issuing an I/O for the same block gets it transparently from CP's in-storage copy. CP has an arbiter to optimize the use of central and/or expanded storage for MDC and paging.

Up through VM/ESA Release 1.2.1, MDC was limited to 4KB-formatted CMS mini-disks and cacheing was done in expanded storage only. Since VM/ESA Release 1.2.2, MDC is enhanced to support any mini-disk (guest or CMS), and can use both central and expanded storage.

SFS DATA SPACES

An SFS directory can be mapped to a VM data space (we'll abbreviate this technique to SFS-DS). For our discussion, SFS directories are similar to mini-disks – you typically ACCESS them before using the files. However, the SFS files are stored on the mini-disks owned by the SFS server. If an SFS directory is associated with a VM data space, the SFS server shares the data space with any user referencing the files. This means that the transmission of file information is no longer over APPC/VM path between the SFS server and the CMS client, but is directly available in virtual storage. When the data space is created, the data blocks on the SFS mini-disks are mapped to page frames in the data space. The SFS server itself will not read the data into the data space but, when the user references a file, CP will use its high-performing paging routines to get the referenced data blocks from the SFS mini-disks into the data space. Other users referencing the same file refer to the same data space pages, so effectively sharing storage.

VIRTUAL DISKS IN STORAGE

Virtual disks in storage (V-disks) were introduced by VM/ESA Release 1.2.1. A V-disk is a mini-disk emulated in CP virtual storage (and thus can be paged out). Virtual disks in storage behave as fast 9336 FBA disks. CP creates them in an ESA/370 data space. When the system goes down, the data is lost. Virtual disks in storage can be shared between virtual machines. They are accessed using any I/O method supported by VM (SIO or SSCH for guests, Diagnose or BLOCKIO for CMS).

Note that the pages in real storage being used by saved segment, shared data spaces, and V-disks are considered to be 'shared storage'. This means that these pages are selected for paging out later than other pages, regardless of how many virtual machines actually use them.

COMPARING THE TECHNIQUES

We will now cover the performance aspects for FSTs, programs, data files, REXX EXECs, etc. In general, performance can be improved by avoiding I/O and/or minimizing paging via storage sharing. But is a virtual disk in storage equivalent to a saved segment? Is a data space a winner? Do saved segments perform better than MDC?

We will start with programs. REXX EXECs are covered later because, to computers, REXX EXECs are ordinary data files that get read and handled by a real program, namely the REXX interpreter. CSP applications have a similar behaviour, whereas compiled REXX EXECs are a special case of modules.

THE BEST TECHNIQUE FOR PROGRAMS

One aspect to bear in mind is that, when a program executes, not all of its subroutines will necessarily be executed. Exception or error routines are examples of this. Loading them in storage is thus a form of overhead that can only be avoided with some of the described techniques.

For program products designed to use saved segments, the choice is clear – you have to use them if you care about performance. Even if there is only one user, there is a gain – because only referenced parts of the program will be paged in.

For MODULEs, you have a choice between:

- Leaving them on a mini-disk (and hoping for MDC benefits).
- Storing them in an SFS-DS .
- Copying them to a virtual disk in storage shared among all users.
- NUCXLOADing them (if reusable).
- Placing them in an LSEG (if re-entrant).

What are the pros and cons of each alternative? In the case of leaving them on a mini-disk, performance is improved when the MDC has a high hit ratio. When CMS reads the program, the I/O will be avoided if CP still has a copy in the MDC. However:

- The running program itself is not shared, so each user of the program has a separate copy in private storage.
- The whole program must be read, including exception routines (it's likely that these will be selected for page-out later because of lack of reference).
- CP will not keep the program in MDC if it is not started frequently. CP tends to cache what is read frequently, and a program is only read when started.

With an SFS-DS, the CMS file containing the program may be in real storage if it is often read. But since the program resides in a data space, and programs can't be executed from data spaces, it must be moved to the user's private address space. So the remarks on leaving them on a mini-disk also apply here.

Storing on a virtual disk in storage is again similar to the SFS-DS solution, but a virtual disk in storage is less practical for this purpose because after each IPL you'd have to copy the programs from a real disk to the virtual disk in storage. Virtual disks in storage are useful for items that don't support SFS (such as VSAM-formatted mini-disks or VSE guest mini-disks).

By NUCXLOADing, you only read the program once and keep it in private storage (from where it can be paged out to expanded storage or to DASD). However, the program isn't shared at all. The NUCXLOAD technique is suitable if the program is exclusive to one

or a few users. For example, the action routines of a PROP should be NUCXLOADed (or EXECLOADed for EXECs) before starting the program.

Only saved segments allow effective sharing of programs among users and, as an extra benefit, only those parts that get executed will be paged in.

Conclusions for programs

The best choice is to use saved segments whenever possible. However, they require more planning and maintenance from the systems programmer, while MDC and SFS-DS are more self-regulating processes.

MDC, SFS-DS, and virtual disks in storage can speed up reading the program from disk, but only if the file is read frequently.

THE BEST TECHNIQUE FOR DATA FILES

Data files can, of course, get the same benefits from MDC and virtual disks in storage as is the case for programs. Using MDC requires minimal effort from the systems programmer.

However, data can be stored in saved segments too, giving the great advantage of sharing. The former 16MB limit explains why the technique was not used frequently in the past.

Normally, programs that read data from shared segments instead of disk have to be specifically designed to do so. However, with CMS Pipelines, reading an EXECLOADed file or reading from disk becomes transparent! Yes, loading a data file in storage with EXECLOAD is fooling CMS, but it works and is supported by CMS Pipelines. Try this, for example:

```
PIPE LITERAL Card 2 | LITERAL Card 1 | > TEST FILE A
EXECLOAD TEST FILE A MYTEST DATA
PIPE < MYTEST DATA | CONSOLE
```

Because placing data in saved segments is so easy, it is worth considering for highly-used data that is not frequently modified.

The very best would be the direct use of VM data spaces but that requires the program to be adapted to use data spaces. You can,

however, indirectly benefit from VM data spaces via the SFS-DS technique, in which case your data processing program doesn't require change. But realize that, compared to direct use of VM data spaces, you then share in 'move mode'. That is, when your program does a read to get data, CMS has to move it from the data space to the program's buffer. Note that with MDC, one also shares storage in 'move mode'.

Improving your EXECs

Note first that, in this discussion, XEDIT macros and Pipeline stages are also 'EXECs' – only the filetype differs – and so are compiled REXX procedures, unless they are compiled into a TEXT object and link-edited into a module.

For sharing and performance aspects, EXECs compare well to programs. So the list of possibilities is very similar:

- Leave them on a mini-disk (and hope for MDC benefits).
- Store them in an SFS-DS.
- Copy them to a virtual disk in storage shared by all users.
- EXECLOAD them.
- Place them in an LSEG or in the CMSINST segment.

You could say that non-compiled EXECs are data to computers, so it must be possible to interpret them directly from a VM data space. Theoretically this is correct, but we've seen that accessing data directly in a data space is not transparent and, for the moment, the CMS REXX interpreter isn't adapted to it. A REXX procedure can, of course, be stored in an SFS-DS, but before execution it will be copied into your address space. For frequently started procedures, the chances are then great that no page-in is required.

Conclusions for EXECs

Place your highly used REXX EXECs in saved segments. Starting with VM/ESA 1.1.0, REXX EXECs can be placed above the 16MB line, relieving the former space constraint. EXEC2 EXECs can also be put into saved segments, but only below 16MB. Although it's clear

that compiled EXECs run much faster, they result in about four times larger files. This means that sharing the coding and avoiding the I/O to load them is even more important once EXECs are compiled.

WHAT ABOUT FSTS?

FSTs can consume a lot of storage. An FST entry for a mini-disk file needs 64 bytes (an entry for an SFS file is a little bigger). Thus, when you ACCESS a mini-disk with 5,000 files, the cost in your address space is over 300KB (75 pages), which all have to be read from disk too. And, even though CMS is clever enough to use hashing techniques to drastically minimize the number of pages to be scanned (two pages per filemode), searching for files is a job CMS has to perform very frequently.

How can we gain performance here?

- Mini-disk cacheing? Although MDC may speed up obtaining the FSTs during ACCESS, it will not help the process of scanning the FSTs. CMS keeps the FSTs in virtual storage so, to find a file, no I/O is required. Since MDC works by eliminating I/Os, it will not help here.
- Saved segments? FSTs fit well into saved segments. The pages will effectively be shared (no moves required), but they must still remain below 16MB. SAVEFD can be used to place FSTs in a 'normal' physical segment, while SEGGEN is the command to place them in an LSEG. The drawback is that, each time something changes on the mini-disk, the segment must be resaved (use 'ACCESS (SAVEONLY' to verify whether a segment is still valid). Remember also that the FSTs for the S and Y disks are saved together with the CMS saved system, so the 19E is a good candidate to receive frequently used files.
- Data spaces? For an SFS-DS, the FSTs do reside in the shared data space and they don't have to be moved to your address space. The extra advantage of an SFS-DS over a CMS mini-disk is that not only are the FSTs shared, but also the files themselves. Furthermore, the FSTs do not consume precious address space below 16MB and they don't have to be manually resaved after files have been updated.

So, saved segments are recommended for FSTs as well. Beware, however, if you often update the disk and resave the segment, because you will end with many copies of the saved segment, thereby reducing the storage effectively shared. In an extreme case, each virtual machine could have its own copy of the saved segment, and nothing is shared any more. How many copies of a segment are acceptable? It depends: for the CMS case, a little calculation seems to indicate that, on a system where some 300 users log-on and back off daily, it is still worthwhile to resave CMS when there are already 10 copies of the CMS segment.

You can use the CP Q NSS USERS segname command to find out how many copies exist and who's using which copy. By restarting users of obsolete class P segments, storage sharing is improved (the CPQUERY EXEC – available on the VM download library – can help you with this task).

HIGHEST ADDRESS

To complete this discussion, we will cover the 'highest' address one should use with saved segments and virtual machine sizes.

In order to describe your address space, CP has to build a so-called 'segment table'. Initially, your segment table has just enough entries to describe your virtual machine size. When you activate a saved segment that was generated at a higher address, CP needs to enlarge your segment table. When you later detach the segment, CP will not downsize your segment table because it thinks that you may use the same segment again later on. Note that since CMS Release 6, saved segments can be loaded inside the virtual machine size (but adding SEGMENT RESERVE commands in the PROFILE or SYSPROF EXEC may be required). And, obviously, the storage occupied by the saved segments cannot be used as private read-write storage.

Knowing this, you should remember that there are three important limits:

- A segment table describing 32MB fits into the base VMDBK (Virtual Machine Description Block) and so has no additional storage cost relative to smaller virtual address sizes.

- From 33MB to 1024MB, the segment table needs one extra 4KB page per user.
- Above 1024MB, CP needs yet another page, and it must be contiguous with the other.

So, if possible, keep virtual machine sizes, and the most commonly used saved segments, below 32MB. If that isn't possible, then what?

- Up to VM/ESA 2.2.0, you can place them very high, but not above 1024MB. The extra real storage cost for a segment at 33MB or at 1024MB is exactly the same – one page per user.
- In VM/ESA 2.3.0, CP became a bit smarter. The unused upper part of segment table pages is reclaimed by CP and used as system 'free' storage.

An example may better illustrate the difference. Suppose a user with a DEF STOR of 32MB loads a 1MB segment located at 511MB. CP fetches a free page, moves the user's segment table inside it, and updates the user's control register seven to reflect the new segment table location and size. It should be clear that half of the segment table page is not used.

Before VM/ESA 2.3.0, that half page was indeed wasted. From 2.3.0 on, CP can use it for free storage.So, from 2.3.0 onwards, for segments that must be placed above 32MB, you gain some space by placing them as low as possible. Don't be overly conservative – if 150 users use a segment at 200MB instead of at 64MB, the extra storage cost is only 32*(200-64)*150 bytes or 640KB.

CONCLUDING GUIDELINES

Data-in-memory techniques can greatly enhance the performance of your system. These techniques eliminate I/Os and, when sharing among users is possible, they reduce real storage consumption.

Saved segments

Even with MDC and VM data spaces, saved segments are invaluable to share programs, EXECs, and mini-disk FSTs.

VM data spaces

An SFS directory in a VM data space performs as well as mini-disks with MDC and shared FSTs. In addition, you get better disk management and you can share more data. Note, however, that SFS file control directories (with full support of aliases, sharing, etc) can't be placed in a VM data space – only directory control directories can.

MDC

MDC is a good performance booster for mini-disks. Note that the CMS mini-disks used by SQL/DS (now known as 'DB2 for VM and VSE') and SFS servers are also eligible for MDC, resulting in an effect of 'bigger buffers' for SFS and SQL/DS.

Note, though, that since VM/ESA 1.2.2, the MDC uses, by default, full-track reads, which is good for sequential access (ie good for most CMS files), but not for random access, such as SQL databases. For SQL/DS, it is best to use the 'SQL Dataspace Feature', or, if using VM/ESA 2.3.0 or 2.2.0+PTF UM28392, use the new 'Record MDC'. The mini-disks used by the SFS catalog (storage pool one) probably also perform better with 'Record MDC'.

Virtual disks in storage

Virtual disks in storage are primarily meant to be shared by VSE guests, used by old CMS applications, or used as a replacement for TDISKs. By old CMS applications, we mean applications that cannot profit from such things as large virtual storage, VM data spaces, or files in an SFS data space. Note, however, that virtual disks in storage are not for free. With the current design, CP considers the pages in use for a virtual disk in storage as shared storage, making them less eligible to be paged out. Hence, a single user with a big and very active virtual disk in storage can take over a big part of central storage.

NUCXLOAD

NUCXLOAD (or EXECLOAD for procedures) is easy to implement and is especially useful when sharing is not important.

CREATING LOGICAL SEGMENTS

We have been saying that saved segments are still the best option for programs, EXECs, and FSTs. Creating logical segments is easy, but many readers may not be familiar with this. Therefore it may be appropriate to mention how LSEGs can be created. For more details, refer to the *VM/ESA Planning and Administration* manual, or have a look in the *VM/ESA Performance* manual.

To create an LSEG:

- Find out what objects you want to place in an LSEG – FSTs, MODULEs, EXECs, XEDIT macros, etc. Apart from FSTs, objects of different types can be placed in an LSEG. A PSEG hosts one or more LSEGs. So, you also have to decide how many LSEGs and PSEGs you'll make. Here we'll suppose you will place your tools (REXX EXECs, XEDIT macros, and some MODULEs) in one LSEG.
- Find a place in real storage to place the segment (remember that LSEGs with FSTs or EXEC2 EXECs must be located below 16MB). Various tools exist to map the storage used by segments:
  - VMFSGMAP, the official VM solution. Issue 'EXEC VMFSGMAP SEGBLD ESASEGS SEGBLIST'. (Because we can't remember that command, we created a SEGMAP EXEC that simply issues the above command.)
  - CPQUERY from the download library.
  - QNSSMAP EXEC, the 'quick and dirty' solution that is appended below.
- Define the segment skeleton. Suppose you found room at 25MB and 1MB is enough, issue:

```
CP DEFSEG mypseg 1900-19FF SR
```

- Define your storage at least 1MB higher than the address of the segment.
- Use XEDIT to create the 'mylseg LSEG' file. Insert lines to describe each object:

```
EXEC STARTXED EXEC     *                    (INSTSEG
```

```
EXEC SUBMIT EXEC       *
EXEC OURPROFL XEDIT    *                    (INSTSEG
EXEC OURFILEL EXEC     *   FILELIST EXEC
MODULE MYBROWSE MODULE *
EXEC SOMEPIPE REXX     *                    (INSTSEG
```

You should note the option INSTSEG. It means that this EXEC is considered to be part of the 'CMS Installation Segment' (CMSINST by default). It influences when the EXEC will be found in the search order:

- Without INSTSEG, the EXEC is considered EXECLOADed, and it will be used even if the user has a copy on his A disk, for example.
- With INSTSEG, when the EXEC will be found depends on the setting of INSTSEG. By default, INSTSEG is 'ON S', which means that this EXEC will be found just before the search of the S disk starts. For example, if a STARTXED EXEC is found on the R disk, the copy in the segment will not be executed, the disk resident version is taken instead.

  So, you must think carefully about the INSTSEG option. EXECMAP can be used to see how often EXECs are executed.
- Use XEDIT to create the 'mypseg PSEG' file. In our case, only one line is required:

  ```
  LSEG mylseg LSEG
  ```
- Access the mini-disks containing your objects, and access CMS resident (MAINT 190) in read/write mode as Z (this way SEGGEN can update the SYSTEM SEGID file immediately).
- Save the PSEG and LSEG(s) by issuing:

  ```
  SEGGEN mypseg PSEG A SYSTEM SEGID Z
  ```
- Now that the LSEG has been created, you must still make your users use it. For FSTs, the ACCESS command will try to use the segment automatically. For other objects, a 'SEGMENT LOAD mylseg (SYSTEM' must be executed. So, you have to include a SEGMENT LOAD in the SYSPROF EXEC, or in another EXEC that your users execute before they use the code you carefully placed in the LSEG.

QNSSMAP EXEC

Here is the quick and dirty, but fast, QNSSMAP EXEC:

```
/* This EXEC creates a simple NSS MAP
      +-----------------------------------+
      | format: |QNNSMAP <ALL>    |
      +-----------------------------------+
 - without option ALL: segment spaces, CMS, and GCS segments not listed
 - with option ALL: everything is listed
/* Don't XEDIT file if disconnected user   */
parse upper source . . myname mytype . syn .
address command
parse upper arg all .
if all='ALL' then           /* All stuff wanted, also CMS NSS-es */
 'PIPE (end ?) CP Q NSS MAP', /* .. so must fill in cols 1-32 and  */
         '|D:  DROP 1',       /* .. 52-61 of all records */
         '|F:  FANOUT',
         '|    SPEC 1-32 1 52.1Ø 33|NFIND _'||,
         '|J:  JUXTAPOSE',
         '|    SPEC 1-32 1 43-* 33 33.1Ø 52 |NFIND _'||,
         '|    XLATE 33-37 A-F FA-FF',
         '|    XLATE 2Ø-2Ø S L',/* SORT S(pace) Before M(ember)*/
         '|    SORT 33-37 2Ø',
         '|    XLATE 33-37 FA-FF A-F',
         '|    XLATE 2Ø-2Ø L S',
         '|T:  FANIN 1 Ø',
         '|    > NSS MAP A',
         '?F:| SPEC 33-* |J:?D:|T:'
else
   'PIPE (end ?) CP Q NSS MAP',
         '|D: DROP 1',
         '|NFIND ___'||,
         '|NLOCATE 15.3 /NSS/',
         '|NLOCATE 15.6 /DCSS-S/',
         '|NLOCATE 15.6 /CPDCSS/',
         '|XLATE 33-37 A-F FA-FF',
         '|SORT 33-37',
         '|XLATE 33-37 FA-FF A-F',
         '|T: FANIN 1 Ø',
         '|> NSS MAP A',
         '?D:|T:'
if rc=Ø then do
   if linesize()>Ø then 'EXEC REXEDIT NSS MAP A NORC'
   else say 'NSS map stored in file NSS MAP A'
end
exit rc
```

Kris Buelens and Guy De Ceulaer
Advisory Systems Engineers
IBM (Belgium)

VM:Secure enhancement rules – part 4

This month we continue the article providing special macros that enhance VM:Secure Rules to allow additional resource access control.

OBJEDIT VMSECURE

```
/* EDIT an Object Rules file */
/* NW */
'TRANSFER OUTPUT  SYSID  USERID'
Pull output  sysid  user
Call Trace output
Arg userid template . '(' xedit_parms
If user = sysid Then Exit -1  /* Don't use on SVM console */

'TEST PROCESS AUTHORIZ $OBJEDIT ANYUSR'
If rc ¬= Ø Then Exit -1
/*************************************************************/
/* Common routine to load the OBJECT settings.               */
/* Variables set:  objcuu         virt dev of object disk */
/*                 objmode        file mode of disk       */
/*                 objdefault     ACCEPT|REJECT default   */
/*************************************************************/
'TEST CMS PIPE (name OBJCLOAD)',
  '< OBJECT SETTINGS |',
  'VAR OBJSET'
If Symbol('OBJSET') ¬= 'BAD' Then Interpret objset
If Symbol('OBJDEFAULT') = 'BAD' Then Do
  'TEST FORMAT EMSG 7ØØØE'
  Exit 299
  End
/*************************************************************/
/* Common routine to check availability of OBJECT RULES    */
/*************************************************************/
'TEST CMS STATE OBJECTS LOCKED' objmode
If rc = Ø Then Do
  'TEST FORMAT EMSG 7ØØØE'
  Exit 299
  End
/*************************************************************/

If userid = '' Then Do
  'TEST FORMAT EMSG Ø38E'
  Exit 24
  End
userobj = userid 'OBJECTS' objmode
lockname = objmode 'OBJECTS' userid
```

```
userXedit = userid 'OBJECTS AØ'
workfile = userid 'CMSUT1' objmode
templatefile = template 'OBJECTS' objmode
defaultfile = 'OBJECT TEMPLATE' objmode
use_templatefile = Ø
'TEST CMS STATE' userobj
If rc ¬= Ø Then Do
  If template = '' Then templatefile = defaultfile
  'TEST CMS STATE' templatefile
  If rc ¬= Ø Then Do
    'TEST FORMAT EMSG 8ØØ3E User OBJECT' template
    Exit 28
    End
  use_templatefile = 1
  End
'TEST PROCESS AUTHORIZ $OBJEDIT' userid
If rc ¬= Ø Then Do
  'TEST FORMAT EMSG 265E OBJEDIT' userid
  Exit 1Ø
  End
'TEST USER EXECUTE ERASE' userXedit
If rc = 35Ø Then Exit 1ØØ
Else If rc = 36 Then Do
  'TEST FORMAT EMSG Ø38ØE'
  Exit 12
  End
'TEST LOCK COND PRIVATE DISK' lockname
If rc ¬= Ø Then Do
  'FORMAT EMSG 364E' userobj
  Exit 14
  End
If use_templatefile Then Do
  'TEST CMS COPYFILE' templatefile userobj
  crc = rc
  If rc ¬= Ø Then Do
    'TEST CMS ERASE' userobj
    'FORMAT EMSG 621E' crc 'COPYFILE'
    'LOCK CLEAR DISK' lockname
    Exit 16
    End
  End
'TEST USER COPYTO' userobj userXedit
If rc ¬= Ø Then Do
  'TEST USER EXECUTE ERASE' userXedit
  'LOCK CLEAR DISK' lockname
  'FORMAT EMSG Ø99I OBJEDIT'
  Exit 1ØØ
  End
Xedit:
'USER STACK LIFO CMS ERASE' userXedit
'TEST USER EXECUTE XEDIT' userXedit '(' xedit_parms
```

```
If rc ¬= Ø Then Do
  'FORMAT EMSG 325E' rc userXedit
  'TEST USER EXECUTE DESBUF'
  If use_templatefile Then 'TEST CMS ERASE' userobj
  'LOCK CLEAR DISK' lockname
  Exit 22
  End
'TEST USER EXECUTE STATE' userXedit
If rc ¬= Ø Then Do
  Call NoChange
  'LOCK CLEAR DISK' lockname
  Exit Ø
  End
'TEST USER COPYFROM' userXedit workfile
crc = rc
If crc ¬= Ø Then Do
  'TEST CMS ERASE' workfile
  If use_templatefile Then 'TEST CMS ERASE' userobj
  'TEST USER EXECUTE ERASE' userXedit
  'LOCK CLEAR DISK' lockname
  'FORMAT EMSG 621E' crc 'COPYFROM'
  Exit 17
  End
'TEST EXEC OBJLOAD' userid
loadrc = rc
If rc = Ø Then Do
  'TEST CMS ERASE' userobj
  'TEST CMS RENAME' workfile userobj
  End
Else If rc = 24 Then Do Forever
  'FORMAT EMSG 469I'
  'TEST FORMAT PROMPT 4Ø4R'
  If rc ¬= Ø Then Call NoChange
  Pull ans .
  If ans = 'YES' Then Do
    'TEST CMS ERASE' workfile
    Signal XEDIT
    End
  Else If ans = 'NO' Then Do
    Call NoChange
    loadrc = Ø
    Leave
    End
  Else 'FORMAT EMSG 431E' ans
  End
Else Call NoChange
'TEST USER EXECUTE ERASE' userXedit
'LOCK CLEAR DISK' lockname

Exit loadrc
NOCHANGE:
```

```
If use_templatefile Then 'TEST CMS ERASE' userobj
'TEST CMS ERASE' workfile
'FORMAT EMSG 8Ø12I'
Return
```

OBJEND VMSECURE

```
/* End all that is...  */
/* NW */

'TRANSFER OUTPUT  SYSID  USERID'
Pull output  sysid  user
Call Trace output
If user ¬= sysid Then Exit -1       /* Only SVM can execute */
'TEST USER LOGOP VMXOBJRULES *ERROR* VM:Secure OBJECT RULES activation
failure.'
'TEST PROCESS SPAWN END FORCE'
'TEST PROCESS READY END FORCE'
'TEST PROCESS SWITCH'
```

OBJFOR VMSECURE

```
/* Check the access allowed for a particular user and OBJECT */
/* NW */

'TRANSFER OUTPUT  SYSID  USERID'
Pull output  sysid  user
Call Trace output
Arg foruser objname object_tokens
'TEST PROCESS AUTHORIZ $OBJFOR' user
If rc ¬= Ø Then Exit -1
If foruser = '' Then Do
  'TEST FORMAT EMSG Ø38E'
  Exit 24
  End
If objname = '' Then Do
  'TEST FORMAT EMSG 8ØØ6E'
  Exit 6
  End
/*********************************************************/
/* Common routine to load the OBJECT settings.           */
/* Variables set:  objcuu         virt dev of object disk */
/*                 objmode        file mode of disk       */
/*                 objdefault     ACCEPT|REJECT default   */
/*********************************************************/
'TEST CMS PIPE (name OBJCLOAD)',
  '< OBJECT SETTINGS |',
  'VAR OBJSET'
If Symbol('OBJSET') ¬= 'BAD' Then Interpret objset
```

```
If Symbol('OBJDEFAULT') = 'BAD' Then Do
  'TEST FORMAT EMSG 7000E'
  Exit 299
  End
/*************************************************************/
/* Common routine to check the availablity of OBJECT RULES.*/
/*************************************************************/
'TEST CMS STATE OBJECTS LOCKED' objmode
If rc = 0 Then Do
  'TEST FORMAT EMSG 7000E'
  Exit 299
  End
/*************************************************************/
object_tokens = Space(object_tokens)
quiet = Abbrev('QUIET',quietopt,1)
'TEST CMS STATE' objname 'OBJDEF' objmode
If rc ¬= 0 Then Do
  'TEST FORMAT EMSG 8200E' objname
  Exit 28
  End

If object_tokens = '',
   | Pos('*',object_tokens) > 0 ,
   | Pos('%',object_tokens) > 0 Then Do
  'TEST FORMAT EMSG 8201E' objname
  Exit 2
  End

'TEST CMS PIPE <' objname 'RULEDEF | VAR OBJDEF'
If Symbol('OBJDEF') ¬= 'BAD' Then Interpret objdef
Else Do
  'TEST FORMAT EMSG 8202E' rc objname 'RULEDEF'
  Exit 300
  End

If tokens.objname ¬= Words(object_tokens) Then Do
  'TEST FORMAT EMSG 8206E' objname tokens.objname
  Exit 4
  End

If default_action.objname ¬= '' Then
  objdefault = default_action.objname

select = objname||'FF'x||Left(object_tokens,1)
findwild = objname||'FF'x||'*'
lookfor = Translate(objname object_tokens,'FF'x,' ')
access_allowed = ''
universal_found = ''

'TEST CMS STATE SYSTEM OBJECTS' objmode
If rc = 0 Then Do
  'TEST CMS PIPE (ENDCHAR ?)|',
```

```
      '< SYSTEM USEROBJ |',
      'DROP 1 |',
      'A: FIND' select'|',
      'STEM SEARCH. |',
      'FIND' lookfor'_|',
      'VAR FOUND',
      '? A: |',
      'FIND' findwild'|',
      'VAR WILD'
  If found ¬= 'FOUND' Then Do
    access_allowed = Word(found,Words(found))
    universal_found = 'EXACT'
    End
  Else Do
    If wild = 'WILD' Then wild = ''
    If search.Ø > Ø | wild ¬= '' Then Do
      Parse Value FEntry() With syskey sysaccess sysmatch
      If syskey ¬= 'NOMATCH' Then Do
        universal_found = syskey
        access_allowed = sysaccess
        End
      End
    End
  End
'TEST CMS STATE' foruser 'OBJECTS' objmode
If rc = Ø Then Do
  'TEST CMS PIPE (ENDCHAR ?)|',
      '<' foruser 'USEROBJ |',
      'DROP 1 |',
      'A: FIND' select'|',
      'STEM SEARCH. |',
      'FIND' lookfor'_|',
      'VAR FOUND',
      '? A: |',
      'FIND' findwild'|',
      'VAR WILD'
  If found ¬= 'FOUND' Then
    access_allowed = Word(found,Words(found))
  Else Do
    If universal_found ¬= 'EXACT' Then Do
      If wild = 'WILD' Then wild = ''
      If search.Ø > Ø | wild ¬= '' Then Do
        Parse Value FEntry() With usrkey usraccess usrmatch
        If usrkey ¬= 'NOMATCH' Then
          If (universal_found usrkey = 'PATTERN PATTERN') |,
             (universal_found usrkey = 'WILDCARD WILDCARD' &,
              Length(usrmatch) >= Length(sysmatch)) Then
            access_allowed = usraccess
        End
      End
    End
```

```
  End
If access_allowed = '' Then access_allowed = objdefault
If access_allowed = 'ACCEPT' Then Do
  'TEST USER MSG' foruser 'access would be ACCEPTED for' object_tokens
  erc = Ø
  End
Else Do
  'TEST USER MSG' foruser 'access would be REJECTED for' object_tokens
  erc = 298
  End
Exit erc
/*************************************************************/
FENTRY: Procedure Expose objname object_tokens search. wild

If wild ¬= '' Then pipestream = 'VAR WILD | STEM SEARCH. |'
Else pipestream = 'STEM SEARCH. |'
'TEST CMS PIPE(endchar ? name FENTRY)|',
   pipestream,
   'A: LOCATE 1-* /%/|',
   'B: FANIN |',
   'CHANGE 1-* /'||'FF'x||'/ /|',
   'SPECS W 2-* 1 |',
   'STEM SEARCH.',
'? A: |',
   'LOCATE 1-* /*/|',
   'SORT DESCENDING|',
   'B:'

If search.Ø = Ø Then Return 'NOMATCH'
tokenwords = Words(object_tokens)
matched_on = 'WILDCARD'
matchtok = ''

Do i = 1 to search.Ø
  match = 1
  Do t = 1 to tokenwords
    token = Word(search.i,t)
    searchtoken = Word(object_tokens,t)
    tokenlen = Length(searchtoken)
    wildcard = Pos('*',token)
    pattern = Pos('%',token)
    If WordPos('Ø',pattern wildcard) > Ø Then
      minchk = Max(pattern,wildcard)-1
    Else minchk = Min(pattern,wildcard)-1
    If Left(token,minchk) ¬== Left(searchtoken,minchk) Then Do
      match = Ø
      Leave t
      End
    Select
      When pattern > Ø & Length(token) ¬= tokenlen &,
           wildcard = Ø Then Do
```

```
        match = Ø
        Leave t
        End
      When pattern > Ø Then Do
        matched_on = 'PATTERN'
        Do While pattern > Ø
          searchtoken = Overlay('%',searchtoken,pattern)
          pattern = Pos('%',token,pattern+1)
          End
        If wildcard = Ø & searchtoken ¬= token Then Do
          match = Ø
          Leave t
          End
        If wildcard > Ø & ¬Check_WildCard(token,searchtoken) Then Do
          match = Ø
          Leave t
          End
        matchtok = matchtok token
        End
      When wildcard > Ø Then Do
        matched_on = 'WILDCARD'
        If ¬Check_WildCard(token,searchtoken) Then Do
          match = Ø
          Leave t
          End
        matchtok = matchtok token
        End
      Otherwise If token ¬= searchtoken Then Do
          match = Ø
          Leave t
          End
        Else Do
        matchtok = matchtok token
        End
      End
    End
  If match Then Do
    Return matched_on Word(search.i,Words(search.i)) Strip(matchtok)
    End
  End
Return 'NOMATCH'
/************************/
CHECK_WILDCARD: Procedure
Arg token , searchtoken
wildcard = Pos('*',token)
If wildcard = Length(token) Then Do
  wildcard = wildcard - 1
  If Left(searchtoken,wildcard) == Left(token,wildcard) Then Return 1
  Return Ø
  End
Else Do While Pos('*',token) > Ø
```

```
  Parse Value token With firstpart '*' . '.' token
  len = Length(firstpart)
  Parse Value searchtoken With srchfirst +(len) . '.' searchtoken
  If firstpart = '' Then Return 1      /* For "xxx*.*" entries */
  If firstpart ¬== srchfirst Then Return Ø
  End
If token ¬= '' & token ¬== searchtoken Then Return Ø
Return 1
```

OBJLOAD VMSECURE

```
/* Load USER OBJECT files */
/* NW */

'TRANSFER OUTPUT  SYSID  USERID'
Pull output  sysid  user
Call Trace output
'TEST PROCESS AUTHORIZ $OBJLOAD ANYUSR'
If rc ¬= Ø Then Exit -1

/*************************************************************/
/* Common routine to load the OBJECT settings.               */
/* Variables set:  objcuu         virt dev of object disk */
/*                 objmode        file mode of disk       */
/*                 objdefault     ACCEPT|REJECT default   */
/*************************************************************/
'TEST CMS PIPE (name OBJCLOAD)',
  '< OBJECT SETTINGS |',
  'VAR OBJSET'
If Symbol('OBJSET') ¬= 'BAD' Then Interpret objset
If Symbol('OBJDEFAULT') = 'BAD' Then Do
  'TEST FORMAT EMSG 7ØØØE'
  Exit 299
  End
/*************************************************************/
objdefloaded. = Ø
default. = ''
/*************************************************************/

Arg loadwho . '(' loadopt .
'TEST PROCESS AUTHORIZ $OBJLOAD' loadwho
If rc ¬= Ø Then Do
  'TEST FORMAT EMSG 265E OBJLOAD' loadwho
  Exit 11
  End
If loadwho = '*' Then Do
  If user ¬= sysid Then Exit -1  /* Only SVM allowed */
  loadwho = '*ALL*'
  'TEST CMS PIPE(name LOADOBJ)|',
     'COMMAND LISTFILE * OBJECTS' objmode '|',
```

```
     'STEM FILE.'
  ten_percent = file.0%10
  tell_at = Format(ten_percent,,0)
  told = 1
 'TEST CMS EXECDROP * USEROBJ'
  Do i = 1 to file.0
    If i = tell_at Then Do
      prct = tell_at/ten_percent*10
      If prct > 100 Then prct = 100
      'TEST FORMAT EMSG 8001I' prct file.0
      told = told + 1
      tell_at = Format(ten_percent*told,,0)
      If (tell_at/ten_percent*10 = 100 & i ¬= file.0) |,
         tell_at > file.0 Then tell_at = file.0
      End
    Call Build_Object_Load file.i
    erc = rc
    If erc ¬= 0 Then Do
      'TEST FORMAT EMSG 8005E' erc file.i
      Exit erc
      End
    If i//10 = 0 Then 'TEST YIELD'
    End
  End
Else Do
  userobj = loadwho 'CMSUT1' objmode
  'TEST CMS STATE' userobj
  If rc ¬= 0 Then Do
    'TEST FORMAT EMSG 8003E User OBJECT' loadwho
    Exit 28
    End
  Call Build_Object_Load userobj
  erc = rc
  If erc ¬= 0 Then Do
    'TEST FORMAT EMSG 8005E' erc userobj
    Exit 305
    End
  End
'TEST FORMAT EMSG 8002I User Objects loaded' loadwho
Exit

/*************************************************************/
Build_Object_Load:
Arg fn ft fm .
'TEST CMS PIPE(ENDCHAR ? )|',
   '<' fn ft fm '|',
   'STRIP BOTH |',
   'SPECS RECNO 1 1-* NW |',
   'NLOCATE 12.1 /*/ |',
   'STEM REC.'
```

```
Do r = 1 to rec.Ø
  rec.r = Space(rec.r)
  Parse Value rec.r With recnum acc_rej objname object_tokens
  If WordPos(acc_rej,'ACCEPT REJECT') = Ø Then Do
    'TEST FORMAT EMSG Ø39E' acc_rej
    Call PROCESS_ERROR 24
    End
  If loadopt ¬= 'FAST' Then Do
    If ¬objdefloaded.objname Then Call Load_Object_Def
    Call Validate_Object
    End
  rec.r = acc_rej objname object_tokens
  End

fm = Left(fm,1)'3'
'TEST CMS PIPE(ENDCHAR ? )|',
   'LITERAL /**/ |',
   'APPEND STEM REC. |',
   'CHANGE 8-* / /'||'FF'x||'/|',
   'SPECS W 2 1 W 1 NW |',
   '>' fn 'LOAD' fm
If loadwho ¬= '*ALL*' Then
  'TEST CMS EXECDROP' fn 'USEROBJ'
'TEST CMS EXECLOAD' fn 'LOAD' fm fn 'USEROBJ'
erc = rc
If erc ¬= Ø Then Do
  'TEST FORMAT EMSG 8ØØ5E' erc fn 'LOAD' fm
  erc = 3Ø5
  End
Return erc

/*************************************************************/
Load_Object_Def:
'TEST CMS STATE' objname 'OBJDEF' objmode
If rc ¬= Ø Then Do
  'TEST FORMAT EMSG 82ØØE' objname
  Call PROCESS_ERROR 24
  End

'TEST CMS PIPE <' objname 'RULEDEF | VAR OBJDEF'
If Symbol('OBJDEF') ¬= 'BAD' Then Interpret objdef
Else Do
  'TEST FORMAT EMSG 82Ø2E' rc objname 'RULEDEF'
  Call PROCESS_ERROR 299
  End
objdefloaded.objname = 1
Return Ø

/*************************************************************/
Validate_Object:
```

```
If object_tokens = '' Then Do
  'TEST FORMAT EMSG 820I1E' objname
  Call PROCESS_ERROR 24
  End

numtokens = Words(object_tokens)

If numtokens < tokens.objname Then Do
  Do t = numtokens+1 to tokens.objname
    If default.t.objname ¬= '' Then
      object_tokens = object_tokens default.t.objname
    Else Do
      'TEST FORMAT EMSG 820I4E' t objname
      Call PROCESS_ERROR 24
      End
    End
  End
Else If numtokens > tokens.objname Then Do
  'TEST FORMAT EMSG 820I3E' objname tokens.objname
  Call PROCESS_ERROR 24
  End

Do t = 1 to tokens.objname
  check = Word(object_tokens,t)
  length = Length(check)
  If check ¬= '*' Then Do
    If length > tokenmax.t.objname Then Do
      'TEST FORMAT EMSG 8I19E word' t ,
            'more max' tokenmax.t.objname
      Call PROCESS_ERROR 24
      End
    If length < tokenmin.t.objname Then Do
      'TEST FORMAT EMSG 8I19E word' t ,
            'less min' tokenmin.t.objname
      Call PROCESS_ERROR 24
      End
    tokenlist = Translate(token.t.objname,' ','|')
    If token.t.objname ¬= '' &,
       WordPos(check,tokenlist) = I Then Do
      'TEST FORMAT EMSG 8I20E word' t
      'TEST FORMAT EMSG 8I22I' tokenlist
      Call PROCESS_ERROR 24
      End
    End
  End
Return I

/*********************************************************/
PROCESS_ERROR:
Arg erc .
```

```
'TEST FORMAT EMSG Ø56I',
   recnum Translate(fn ft fm,'ØØ'x,' ')
Exit erc
```

OBJLOCK VMSECURE

```
/* Psuedo Lock/Unlock access to the OBJECT RULES */
/* NW */

'TRANSFER OUTPUT  SYSID  USERID'
Pull output  sysid  user
Call Trace output

If user ¬= sysid Then Exit -1       /* Only SVM allowed to issue */
Arg objmode . '(' opt .
If objmode = '' Then Do
  /*****************************************************************/
  /* Common routine to load the OBJECT settings.                   */
  /* Variables set:  objcuu                virt dev of object disk */
  /*                 objmode               file mode of disk       */
  /*                 objdefault            ACCEPT|REJECT default   */
  /*****************************************************************/
  'TEST CMS PIPE (name OBJCLOAD)',
    '< OBJECT SETTINGS |',
    'VAR OBJSET'
  If Symbol('OBJSET') ¬= 'BAD' Then Interpret objset
  If Symbol('OBJDEFAULT') = 'BAD' Then Do
    'TEST FORMAT EMSG 7ØØØE'
    Exit 299
    End
  End
 If opt = '' Then Do
  'TEST CMS PIPE VAR OBJSET | > OBJECTS LOCKED' objmode
  'TEST CMS STATE OBJECTS LOCKED' objmode
  'TEST USER MSG LOCKED on disk' objmode rc
  'TEST FORMAT EMSG 7ØØ2I LOCKED.'
  End
Else If opt = 'CLEAR' Then Do
  'TEST CMS ERASE OBJECTS LOCKED' objmode
  'TEST USER MSG UNLOCKED on disk' objmode
  'TEST FORMAT EMSG 7ØØ2I UNLOCKED.'
  End
Exit Ø
```

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

James S Vincent
Software Specialist
Nationwide Insurance (USA)

A full screen console interface – part 12

Editor's note: the following article is an extensive piece of work which will be published over several issues of VM Update. *It was felt that readers could benefit from the entire article and from the individual sections. Any comments or recommendations would be welcomed and should be addressed either to Xephon or directly to the author at fernando_duarte@vnet.ibm.com.*

CSCUOP ASSEMBLE

This module adds support for the OP command. It allows you to operate a controlled DSM. The format is 'OP x comm', where 'x' is the DSM prefix as defined in the configuration file and 'comm' is the command to send. You must have class 06 and the DSM class. In the following example CMS001 is allowed to operate RSCS but not VTAM:

```
USER   CMSØØ1   Classes Ø1 Ø2 Ø3 Ø4 Ø5 Ø6   25
PREFIX R RSCS   Class                      25   Blue
PREFIX V VTAM   Class                      26   Pink
```

Code

```
         TITLE 'CSCUOP - CSC Process User OP command'
CSCUOP   START X'Ø1E5Ø8'
         PRINT NOGEN
         CSCHDR                        User OP command
*
* Process OP command
*
         USING UIDSECT,R8              UID (user) Block
         USING PFXSECT,R2              PFX Prefix table
         SPACE
         LA    RØ,UOPTABLE             Address table to search
         GO    CSCSCN                  Do it
         BNZ   UOP6ØØ                  Nothing found, that's bad news
         LTR   R15,R15                 Is it USer ...?
         BZ    UOP2ØØ                  No, better be a valid prefix
         SR    RØ,RØ                   Not more tables to look up
         GO    CSCSCN                  Get user-id
         BNZ   UOP62Ø                  Not there, more bad news
         LA    RØ,8                    Maximum for user-id length is 8
         CR    RØ,R1
```

```
         BL    UOP640                  Too long
         MVC   UOPUSER,SCANUPP         Save user-id for now
         L     R1,PFXPTR               Address Prefix table
UOP100   LTR   R2,R1                   Check for End-Of-Table
         BZ    UOP660                  User not on prefix table
         L     R1,PFXFWD               Address next entry
         CLC   PFXUSER,UOPUSER         Compare users
         BNE   UOP100                  Not this one
         B     UOP400                  Validate user-id
         SPACE
UOP200   LA    R0,1                    Maximum length for Prefix is one
         CR    R0,R1
         BL    UOP700                  Too long
         L     R1,PFXPTR               Search Prefix table
UOP300   LTR   R2,R1                   Check for End-Of-Table
         BZ    UOP720                  Not found, too bad
         L     R1,PFXFWD               Address next entry
         CLC   PFXPREF,SCANUPP         Compare prefixes
         BNE   UOP300                  Not this one
         MVC   UOPUSER,PFXUSER         Prefix found, copy user-id
UOP400   SR    R1,R1                   Validate user
         IC    R1,PFXCLASS             Get destination user class
         O     R1,UIDCLASS             Match with user classes
         C     R1,UIDCLASS
         BNE   UOP800                  User not authorized
UOP450   BAS   R14,SEND
         B     UOP900
         DROP  R2
         SPACE
UOP600   MSG   0310,USER               Missing prefix or user-id
         B     UOP900
         SPACE
UOP620   MSG   0370,USER               Missing user-id value
         B     UOP900
         SPACE
UOP640   MSG   0371,USER               User-id too long
         B     UOP900
         SPACE
UOP660   MSG   0372,USER               User-id not defined
         B     UOP900
         SPACE
UOP700   MVC   2(L'DOTS,R6),DOTS       Prefix not one byte long
         MSG   0373,USER
         B     UOP900
         SPACE
UOP720   MSG   0374,USER               Prefix not defined
         B     UOP900
         SPACE
UOP800   LA    R6,UOPUSER              User not authorized
         MSG   0375,USER
*        B     UOP900
```

```
         SPACE
UOP900   BACK
         SPACE 3
*
* Log user command and call CP to forward it to destination user-id
*
SEND     EQU   *
         ST    R14,SENDSV14
         MVC   OPCOMM(L'UOPUSER),UOPUSER Copy destination to CP SEND
         LA    R4,OPCOMM+L'UOPUSER     Address end of user-id
SEND100  BCTR  R4,0
         CLI   0(R4),C' '              Remove trailing blanks
         BE    SEND100
         MVI   1(R4),C' '              Keep one single blank
         LA    R4,2(,R4)               Address to move user command
         A     R6,SCANLEN              Skip prefix or user-id
         LA    R6,1(,R6)               Allow one space before command
         L     R1,CSCBUFFE             Address end of input data
         SR    R5,R5                   Possible length of user command
         CR    R1,R6                   Anything entered by the user
         BNH   SEND200                 No, user just wants press ENTER
         SR    R1,R6                   Length of input command
         LR    R5,R1                   Copy to R5
         BCTR  R1,0                    Prepare to EXecute
         EX    R1,SENDMVC1             Move user command to CP area
SEND200  LA    R1,0(R5,R4)             End address for CP command
         LA    R0,OPSEND               Begin address
         SR    R1,R0                   Calculate length
         ST    R1,OPLEN                Store it
         LA    R0,OPSEND               Address message area
         L     R2,OPLEN                Load message length
         O     R2,UOPRESP              Request CP response in buffer
         LA    R1,CSCBUFF              Address response buffer
         LA    R3,L'CSCBUFF            Buffer length
         DIAG  R0,R2,X'0008'           Call CP to execute command
         LTR   R3,R3                   Any error message
         BZ    SEND300                 No, log command
         ST    R1,SCRMSG               Yes, store address
         BCTR  R3,0                    Do not display CP end NL (X'15')
         ST    R3,SCRMSGL              Store also length
         OI    UIDOPT4,UIDBMSG+UIDBALM Display message and beep beep
         B     SEND900
         SPACE
SEND300  MVC   CSCBUFF(8),UOPUSER      Destination user-id
         LA    R2,CSCBUFF+8            Prepare to log record
         MVC   0(L'UOPIDBEG,R2),UOPIDBEG Move log record id <CSC ...
         LA    R2,L'UOPIDBEG(,R2)      Adjust pointer
         MVC   0(L'UIDVMID,R2),UIDVMID Move originating user-id
         LA    R2,L'UIDVMID(,R2)       Adjust pointer
SEND400  BCTR  R2,0
```

```
         CLI   Ø(R2),C' '              Remove trailing blanks
         BE    SEND4ØØ
         LA    R2,1(,R2)
         MVC   Ø(L'UOPIDEND,R2),UOPIDEND End log record id >: ...
         LA    R2,L'UOPIDEND(,R2)
         LTR   R5,R5                   Check user command length
         BZ    SEND5ØØ                 Nothing...
         EX    R1,SENDMVC2             Move user command
SEND5ØØ  LA    R1,Ø(R5,R2)             Address end of log record
         ST    R1,CSCBUFFE             Store it for CSCCPW
SEND8ØØ  ST    R8,SENDUID              Save address of our UID block
         GO    CSCCPW                  Log User command on Data File
         L     R8,SENDUID              Restore our UID block address
SEND9ØØ  L     R14,SENDSV14
         BR    R14
         SPACE
SENDMVC1 MVC   Ø(*-*,R4),Ø(R6)         Move user command to CP SEND
SENDMVC2 MVC   Ø(*-*,R2),Ø(R4)         Move user command to log record
         SPACE 3
SENDSV14 DS    F                       Save area for SEND R14
SENDUID  DS    F                                    UID block
         SPACE
UOPTABLE CMMD  (B,ØØ,Ø2,USER,*)        OP command options
         SPACE
UOPUSER  DS    CL8
UOPRESP  DC    X'40000000'             Request CP response in buffer
DOTS     DC    C'... '                 Make some message look "nice"
UOPIDBEG DC    C'<CSC '                Log record begin id (eye catch)
UOPIDEND DC    C'>: '                  Log record end id
         SPACE
OPLEN    DS    F                       Length of CP SEND command
OPSEND   DC    C'SEND '                Op command
OPCOMM   DS    CL8Ø                    Actual user command
         SPACE
         CSCDATA
         CSCDS (UID,PFX)
         REGEQU
         END
```

CSCUEX ASSEMBLE

This module adds support for the INCLUDE and EXCLUDE commands. These allow you to select the messages to display. INCLUDE R would display only messages from DSMs with the prefix R.

```
         TITLE 'CSCUEX - CSC Process User Exclude/Include commands'
CSCUEX   START X'Ø1DD7Ø'
```

```
         PRINT NOGEN
         CSCHDR                         User Exclude command
*
* Process EXCLUDE command
*
         USING UIDSECT,R8               UID (user) Block
         USING PFXSECT,R1               PFX (prefix) Block
         SPACE
EXCLUDE  EQU   *                        EXCLUDE command
         AR    R6,R1                    Skip command name
EXC100   LA    R6,1(,R6)                Advance pointer
         C     R6,CSCBUFFE              Check for end of data
         BNL   EXC800                   Nothing found, reset options
         CLI   0(R6),C' '               Check for first non-blank
         BE    EXC100                   Not yet, loop back
EXC200   OI    0(R6),X'40'              Dirty uppercase
         LA    R1,PFXPTR                Address Prefix table
EXC300   L     R1,PFXFWD                Get next entry address
         LTR   R1,R1                    Is it valid?
         BZ    EXC700                   No, the prefix was invalid
         CLC   PFXPREF,0(R6)            Check prefix
         BNE   EXC300                   Not this one, try another
         TM    UIDOPT2,UIDEXC           Is Exclude option set?
         BO    EXC400                   Yes, add new prefix
         TM    UIDOPT2,UIDINC           Is Include option set?
         BO    EXC500                   Yes, remove new prefix
         MVC   UIDSEL,BLANKS            Nothing set, clear field
         OI    UIDOPT2,UIDEXC           Set Exclude option
EXC400   BAS   R14,ADDPREF              Add new prefix
         BZ    EXC900                   Check all prefixes
         B     EXC600                   Unable to add prefix
         SPACE
EXC500   BAS   R14,DELPREF              Delete new prefix
         BZ    EXC900                   Did it work?
EXC600   MSG   0380,USER                No space in UIDSEL to add,
         B     EXC900                      duplicate or not found
         SPACE
EXC700   MSG   0381,USER                Invalid prefix, not on PFX table
         B     EXC900                   Keep going, check all input
         SPACE
EXC800   NI    UIDOPT2,X'FF'-UIDINC-UIDEXC Reset Include and Exclude
EXC900   LA    R6,1(,R6)                Advance pointer
         C     R6,CSCBUFFE              Anything left in the buffer
         BNL   EXC910                   No, all done...
         CLI   0(R6),C' '               Yes, skip spaces
         BE    EXC900
         B     EXC200                   Process everything else
         SPACE
EXC910   CLC   UIDSEL,BLANKS            Any prefix left?
         BE    EXC920                   No, reset options
```

```
         BAS   R14,SRTPREF             Yes, sort them
         B     EXC99Ø
         SPACE
EXC92Ø   NI    UIDOPT2,X'FF'-UIDINC-UIDEXC Reset options
EXC99Ø   OI    UIDOPT4,UIDBHDR         Remember to refresh Header line
         NI    UIDOPT4,X'FF'-UIDBSCR   Also rebuild the screen
         BACK
         SPACE 3
*
* Process INCLUDE command
*
CSCUEXIN RELOC                         INCLUDE command
         AR    R6,R1                   Skip command name
INC1ØØ   LA    R6,1(,R6)               Advance pointer
         C     R6,CSCBUFFE             Check for end of data
         BNL   INC8ØØ                  Nothing found, reset options
         CLI   Ø(R6),C' '              Check for first non-blank
         BE    INC1ØØ                  Not yet, loop back
INC2ØØ   OI    Ø(R6),X'4Ø'             Dirty uppercase
         LA    R1,PFXPTR               Address Prefix table
INC3ØØ   L     R1,PFXFWD               Get next entry address
         LTR   R1,R1                   Is it valid?
         BZ    INC7ØØ                  No, the prefix was invalid
         CLC   PFXPREF,Ø(R6)           Check prefix
         BNE   INC3ØØ                  Not this one, try another
         TM    UIDOPT2,UIDINC          Is Include option set?
         BO    INC4ØØ                  Yes, add new prefix
         TM    UIDOPT2,UIDEXC          Is Exclude option set?
         BO    INC5ØØ                  Yes, remove new prefix
         MVC   UIDSEL,BLANKS           Nothing set, clear field
         OI    UIDOPT2,UIDINC          Set Include option
INC4ØØ   BAS   R14,ADDPREF             Add new prefix
         BZ    INC9ØØ                  Check all prefixes
         B     INC6ØØ                  Unable to add prefix
         SPACE
INC5ØØ   BAS   R14,DELPREF             Delete new prefix
         BZ    INC9ØØ                  Did it work?
INC6ØØ   MSG   Ø38Ø,USER               No space in UIDSEL to add,
         B     INC9ØØ                     duplicate or not found
         SPACE
INC7ØØ   MSG   Ø381,USER               Invalid prefix, not on PFX table
         B     INC9ØØ                  Keep going, check all input
         SPACE
INC8ØØ   NI    UIDOPT2,X'FF'-UIDINC-UIDEXC Reset Include and Exclude
INC9ØØ   LA    R6,1(,R6)               Advance pointer
         C     R6,CSCBUFFE             Anything left in the buffer
         BNL   INC91Ø                  No, all done...
         CLI   Ø(R6),C' '              Yes, skip spaces
         BE    INC9ØØ
         B     INC2ØØ                  Process everything else
```

```
         SPACE
INC91Ø   CLC   UIDSEL,BLANKS           Any prefix left?
         BE    INC92Ø                 No, reset options
         BAS   R14,SRTPREF            Yes, sort them
         B     INC99Ø
         SPACE
INC92Ø   NI    UIDOPT2,X'FF'-UIDINC-UIDEXC Reset options
INC99Ø   OI    UIDOPT4,UIDBHDR        Remember to refresh Header line
         NI    UIDOPT4,X'FF'-UIDBSCR  Also rebuild the screen
         BACK
         SPACE 3
*
* Add new prefix to existing list (UIDSEL)
*
ADDPREF  EQU   *                      Add new prefix to UIDSEL
         LA    R2,UIDSEL-1            Prepare to loop
         LA    R3,UIDSEL-1+L'UIDSEL   Address last byte
ADDP1ØØ  LA    R2,1(,R2)              Increment pointer
         CR    R2,R3                  End of field?
         BH    ADDP2ØØ                Yes, no space available
         CLI   Ø(R2),C' '             Is it a blank?
         BE    ADDP3ØØ                Yes, add new prefix
         CLC   Ø(1,R2),Ø(R6)          Is the prefix already there?
         BNE   ADDP1ØØ                No, check all
ADDP2ØØ  LTR   R14,R14                Yes, don't create duplicates
         BR    R14
         SPACE
ADDP3ØØ  MVC   Ø(1,R2),Ø(R6)          Move new prefix
         CR    R14,R14                Generate a zero cc
         BR    R14
         SPACE
*
* Remove prefix from existing list (UIDSEL)
*
DELPREF  EQU   *                      Delete prefix from UIDSEL
         LA    R2,UIDSEL-1            Prepare to loop
         LA    R3,UIDSEL-1+L'UIDSEL   Address last byte
DELP1ØØ  LA    R2,1(,R2)              Increment pointer
         CR    R2,R3                  End of data?
         BHR   R14                    Yes, prefix not found
         CLC   Ø(1,R2),Ø(R6)          Is it this one?
         BNE   DELP1ØØ                No, keep trying
DELP2ØØ  MVC   Ø(1,R2),1(R2)          Yes, shift other prefixes left
         LA    R2,1(,R2)              Advance pointer
         CR    R2,R3                  Everything shifted?
         BNH   DELP2ØØ                No, so do it
         MVI   Ø(R3),C' '             Yes, clear last byte
         CR    R14,R14                Generate a zero cc
         BR    R14
         SPACE
```

```
*
* Sort prefixes from existing list (UIDSEL)
*
*        Note  This routine uses a customized High Performance Special
*              Sort Algorithm capable of sorting UIDSEL in less then
*              twenty milliseconds on a medium size MP computer
*
SRTPREF  EQU   *                       Sort prefixes from UIDSEL
         LA    R1,UIDSEL+l'UIDSEL      Address end of UIDSEL field
SRTP1ØØ  BCTR  R1,Ø                    Go back one byte
         CLI   Ø(R1),C' '              Is it a blank?
         BE    SRTP1ØØ                 Yes, keep trying
         LA    R2,UIDSEL-1             Prepare to loop
SRTP2ØØ  LA    R2,1(,R2)               Address first or next byte
         CR    R2,R1                   Compare with last non-blank
         BER   R14                     All done, return
         LR    R3,R2                   Address byte being tested
SRTP3ØØ  LA    R3,1(,R3)               Address next byte
         CR    R3,R1                   Still a valid address?
         BH    SRTP2ØØ                 No, check other bytes
         CLC   Ø(1,R2),Ø(R3)           Compare bytes
         BL    SRTP3ØØ                 Already in sequence, keep going
         IC    RØ,Ø(,R2)               Swap bytes
         MVC   Ø(1,R2),Ø(R3)
         STC   Ø,Ø(,R3)
         B     SRTP3ØØ                 Sort them all
         SPACE 3
         CSCDATA
         CSCDS (UID,PFX)
         REGEQU
         END
```

CSCUST ASSEMBLE

This module adds support for the SET command. It performs no useful function yet, except to prevent CSCSVP from abending.

```
         TITLE 'CSCUST - CSC Process User Set command'
CSCUST   START X'Ø45678'
         PRINT NOGEN
         CSCHDR                        Set command
*
* Process Set command
*
*        USING IPARML,R9               IUCV Parameter List
*        USING UIDSECT,R8              UID (user) Block
*        USING CCHSECT,R7              CCH (cache) Block
         SPACE
         LA    RØ,SETTABLE
```

```
         GO    CSCSCN
         BNZ   UST8ØØ
 MSG 1233
         B     UST9ØØ
         SPACE
UST8ØØ   MSG   Ø31Ø
*        B     UST9ØØ
         SPACE
UST9ØØ   BACK
         SPACE 3
*
* Process OP subcommand
*
OP       EQU   *                       OP subcommand
 MSG 4455
         B     UST9ØØ
         SPACE 3
SETTABLE CMMD  (I,Ø2,Ø1,'OP         ',OP),       Set Command Options   *
               (I,Ø2,Ø3,'XXXXX      ',*)
OPTABLE  CMMD  (I,Ø2,Ø2,'ON         ',*),        OP options            *
               (I,Ø2,Ø3,'OFF        ',*)
         CSCDATA
*        CSCDS (UID,RDF,PFX,MSG,CCH)
         REGEQU
         END
```

CSCOPC ASSEMBLE

This module is the main entry point for the CSCSVP operator commands. It processes the CMS and END commands. The END command terminates CSCSVP and the CMS command allows you to enter any CMS command without terminating CSCSVP. Note that data collection and user sessions are suspended while the CMS command executes.

```
         TITLE 'CSCOPC - CSC Process Operator Commands (console input)'
CSCOPC   START X'Ø17EF8'
         PRINT NOGEN
         CSCHDR                        Process Operator Commands
*
* Process Console command
*
*
         SR    RØ,RØ
         ST    RØ,SCANLEN              Start new scan
         LA    RØ,OPCTABLE
         GO    CSCSCN                  Scan command name
```

```
         BNZ   OPC1ØØ                  Nothing, display prompt
         LTR   R15,R15                 Is command valid?
         BNZ   OPC2ØØ                  Yes, process it
         MSG   Ø6ØØ                    No, display error message
OPC1ØØ   MSG   Ø6Ø1                    Display CSC prompt message
         B     OPC9ØØ
         SPACE
         USING CMDSECT,R2
OPC2ØØ   MVC   CSCCOMM,CMDNAME         Save command name
         GO                           , Execute processing routine
         LR    R2,R15
         MSG   Ø6Ø2                    Tell user command finished
OPC9ØØ   BACK                          All done, go back
         SPACE
         DROP  R2
         SPACE 3
*
* CMS  Execute any CMS command
*
CMS      EQU   *                       CMS Execute any CMS command
         USING NUCON,RØ
         ST    R14,CMDSV14
         SR    RØ,RØ
         GO    CSCSCN                  No table to search
         BZ    CMS1ØØ                  Something found, process it
         MSG   Ø61Ø,CC                 CMS command is missing
         L     R14,CMDSV14
         BR    R14
         SPACE
CMS1ØØ   IPK                           Insert PSW key into R2
         ST    R2,PSWKEY               Save PSW key temporarily
         SR    R2,R2                   Zero register
         SPKA  Ø(R2)                   Store PSW key of zero
         LR    R1,R6                   Address real CMS command
         L     RØ,CSCBUFFE             End address
         ST    R1,CMSEPLA              Build extended parameter list
         ST    RØ,CMSEPLE
         SR    RØ,R1                   Command length into R1
         L     15,ASCANN               Build tokenized PL for CMS
         BASR  14,15                   Call CMS to do it
         L     R2,PSWKEY               Load previous PSW key
         SPKA  Ø(R2)                   Store it into PSW
         TM    CSCFLGØ1,HNDIOS         Check for Console trap
         BZ    CMS4ØØ
         HNDIO CLR,DEVNAME=CONS        Disable trap
CMS4ØØ   CMSCALL PLIST=CMSPL,EPLIST=CMSEPL,CALLTYP=SUBCOM
         LR    R2,R15
         WAITT                         Wait for I/O to complete
         MSG   Ø611                    Display message end, enable cons
```

```
         B     OPC900                  Do not display CSC msg (0602)
         DROP  R0
         SPACE 3
*
* End, Exit, Goback, Quit, Terminate... and close the shop
*
END      EQU   *                       END command
         ST    R14,CMDSV14
         SR    R0,R0                   No table to search
         GO    CSCSCN                  Any operand?
         BNZ   END100                  No, that's good news
         MSG   0605,CC                 Display error message
         B     END900
         SPACE
END100   MSG   0619                    Tell everybody we are going
         OI    CSCFLG02,WORKEND        Remember to close the shop
         SR    R15,R15                 Nothing can go wrong (almost)
END900   L     R14,CMDSV14
         BR    R14
         SPACE 3
CMDSV14  DS    F                       Save area for input commands
PSWKEY   DS    F                       Save user PSW key
         SPACE
OPCTABLE CMMD  (I,00,03,CMS,CMS),      Operator commands              *
               (I,00,03,END,END),                                     *
               (I,00,03,BYE,END),                                     *
               (I,00,04,EXIT,END),                                    *
               (I,00,06,GOBACK,END),                                  *
               (I,00,04,QUIT,END),                                    *
               (I,00,09,TERMINATE,END),                               *
               (E,00,01,QUERY,CSCOPQ),                                *
               (E,00,03,START,CSCOPA),                                *
               (E,00,04,STOP,CSCOPASP)
         SPACE
CMSPL    DC    C'CMS     '
CMSEPL   DC    A(CMSPL)
CMSEPLA  DS    F
CMSEPLE  DS    F
         DC    F'0'                *4* Extended Parameter List word 4
         SPACE
         LTORG
         SPACE
         CSCDATA
         CSCDS (CMD)
         REGEQU
         NUCON
         END
```

CSCRLS ASSEMBLE

This module is invoked during termination of CSCSVP. It releases all storage allocated and terminates all user sessions.

```
         TITLE 'CSCRLS - CSC Release allocated storage'
CSCRLS   START X'Ø178A8'
         PRINT NOGEN
         CSCHDR                        Release allocated storage
*
* Release allocated storage... terminate all IUCV sessions
*
         USING UIDSECT,R8              UID (user) Block
         L     RØ,TRACESZ          *T* Start with Trace Table (testing)
         L     R1,TRACEBEG         *T*
         LINK  RELEASE
         L     RØ,CACHESZ              Cache size
         L     R1,CACHE
         LINK  RELEASE
         L     R1,PFXPTR               Address Prefix Table
         SPACE
         USING PFXSECT,R1
REL1ØØ   L     R2,PFXFWD               Save address of next entry
         LA    RØ,PFXSIZE              Entry length
         LINK  RELEASE
         LTR   R1,R2                   Do all entries
         BNZ   REL1ØØ
         DROP  R1
         SPACE
         L     R2,MSGPTR               Address Message Table
         SPACE
         USING MSGSECT,R1
REL2ØØ   LTR   R1,R2                   Check for end of table
         BZ    REL3ØØ
         L     R2,MSGFWD               Save address of next entry
         LA    RØ,MSGSIZE              Entry length
         LINK  RELEASE
         B     REL2ØØ                  Do all entries
         DROP  R1
         SPACE
REL3ØØ   L     R2,HLDPTR               Address messages on Hold
         SPACE
         USING CCHSECT,R1
REL31Ø   LTR   R1,R2                   Check messages on Hold
         BZ    REL32Ø
         L     R2,CCHFWD               Save address of next entry
         LA    RØ,CCHSIZE              Entry length
         LINK  RELEASE
         B     REL31Ø                  Do all entries
         DROP  R1
         SPACE
```

```
REL32Ø   L     R2,RTEPTR              Address Route table
         SPACE
         USING RTESECT,R1
REL33Ø   LTR   R1,R2                  Check Route table entries
         BZ    REL34Ø
         L     R2,RTEFWD              Save address of next entry
         LA    RØ,RTESIZE             Entry length
         LINK  RELEASE
         B     REL33Ø                 Do all entries
         DROP  R1
         SPACE
REL34Ø   L     R2,USRPTR              Address User table
         SPACE
         USING USRSECT,R1
REL35Ø   LTR   R1,R2                  Check User table entries
         BZ    REL36Ø
         L     R2,USRFWD              Save address of next entry
         LA    RØ,USRSIZE             Entry length
         LINK  RELEASE
         B     REL35Ø                 Do all entries
         DROP  R1
         SPACE
REL36Ø   L     R2,RNDPTR              Address Resource table
         SPACE
         USING RNDSECT,R1
REL37Ø   LTR   R1,R2                  Check Resource table entries
         BZ    REL4ØØ
         TM    RNDOPT1,RNDOLCL        Local Node has no buffers
         BO    REL38Ø
         TM    RNDOPT1,RNDOTMP        Same for temporary entries
         BO    REL38Ø
         LA    RØ,RNDBUFSZ            RND Send/Receive buffer size
         L     R1,RNDSBUFF
         LINK  RELEASE                Release Send buffer
         LR    R1,R2                  Restore RND pointer
         LA    RØ,RNDBUFSZ
         L     R1,RNDRBUFF
         LINK  RELEASE                Release Receive buffer
         LR    R1,R2                  Restore RND pointer
REL38Ø   L     R2,RNDFWD              Save address of next entry
         LA    RØ,RNDSIZE             Entry length
         LINK  RELEASE
         B     REL37Ø                 Do all entries
         DROP  R1
         SPACE
REL4ØØ   L     R2,TMRPTR              Address Event table
         SPACE
         USING TMRSECT,R1
REL41Ø   LTR   R1,R2                  Check Event table entries
         BZ    REL5ØØ
         L     R2,TMRFWD              Save address of next entry
```

```
         LA    RØ,TMRSIZE              Entry length
         LINK  RELEASE
         B     REL41Ø                  Do all entries
         DROP  R1
         SPACE
REL5ØØ   L     R8,UIDPTR               Pending sessions
         LTR   R8,R8
         BZ    REL6ØØ                  Not found...
REL51Ø   LA    RØ,UIDSIZE
         LR    R1,R8
         L     R8,UIDFWD               Address following UID block
         LINK  RELEASE                 Release UID block
         LTR   R8,R8                   Do all blocks
         BNZ   REL51Ø
REL6ØØ   L     R8,SSSPTR               Active sessions
         LTR   R8,R8
         BZ    REL7ØØ
         L     RØ,UIDPID               Get PATHID (first two bytes)
         GO    CSCSEV                  Sever connection
         C     R8,SSSPTR               Was UID block released?
         BNE   REL6ØØ                  Yes, next
         GO    CSCUSARL                No, release the UID block
         B     REL6ØØ                  Sever all sessions
         SPACE
REL7ØØ   L     R2,RDFPTR               Release all read DF buffers
         USING RDFSECT,R2
REL71Ø   L     R3,RDFFWD               Address next entry
         LA    RØ,512                  Release buffer and RDF Block
         L     R1,RDFADDR
         LINK  RELEASE                 DF buffer
         LA    RØ,RDFSIZE
         LR    R1,R2
         LINK  RELEASE                 RDF block
         LR    R2,R3
         C     R2,RDFPTR               Do all buffers and RDF Blocks
         BNE   REL71Ø
REL8ØØ   L     R2,FSALLDW              Double words allocated
         S     R2,FSRELDW              Double words released
         BZ    REL9ØØ                  Was everything released?
         SLL   R2,3                    No, convert dwords to bytes
         L     R3,FSALL                Number of allocations
         S     R3,FSREL                Subtract releases
         MSG   Ø18Ø                    Display warning message
REL9ØØ   BACK
         SPACE 3
         CSCDATA
         CSCDS (UID,RDF,PFX,MSG,RTE,RND,USR,TMR,CCH)
         REGEQU
         END
```

CSCCLS ASSEMBLE

This module is invoked during termination of CSCSVP. It ends IUCV and I/O interrupt processors.

```
         TITLE 'CSCCLS - CSC CLose the shop'
CSCCLS   START X'Ø17BFØ'
         PRINT NOGEN
         CSCHDR                        Terminate
*
* Terminate IUCV and Console I/O processing
*
*
         TM    CSCFLGØ1,CMSIUCVC       Terminate IUCV session with CP
         BZ    CLS1ØØ
         NI    CSCFLGØ1,X'FF'-CMSIUCVC
         LA    R2,CSCNAME
         CMSIUCV SEVER,NAME=(R2),PRMLIST=(R9)
         LTR   R15,R15
         BZ    CLS1ØØ
         MSG   Ø19Ø,RC
         SPACE
CLS1ØØ   TM    CSCFLGØ1,HNDIUCVS       Make CMS happy
         BZ    CLS2ØØ
         NI    CSCFLGØ1,X'FF'-HNDIUCVS
         LA    R2,CSCNAME
         HNDIUCV CLR,NAME=(R2)
         LTR   R15,R15
         BZ    CLS2ØØ
         MSG   Ø191,RC
         SPACE
CLS2ØØ   TM    CSCFLGØ1,HNDIOS         Restore console processing
         BZ    CLS9ØØ
         NI    CSCFLGØ1,X'FF'-HNDIOS
         HNDIO CLR,DEVNAME=CONS
         LTR   R15,R15
         BZ    CLS9ØØ
         MSG   Ø192,RC
CLS9ØØ   BACK
         SPACE 3
         CSCDATA
         REGEQU
         END
```

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

Fernando Duarte
Analyst (Canada)

IBM has announced VM and VSE versions of its DB2 Forms Version 1.0, for building and distributing application front ends to DB2 workstation databases. Applications can be created by developers, governed by administrators, and run by end users on Windows 95, 98, and NT 3.51 or later.

It's compliant with the Open Group's Distributed Relational Database Architecture (DRDA). Global connectivity is possible between DB2 Forms applications and multiple DB2 database platforms via publicly-accessible Internet connections, dedicated dial-up lines, TCP/IP intranets, or closed SNA environments.

For further information contact your local IBM representative.

* * *

VM users can benefit from SDI's Cache Magic, a software implementation of the DASD cacheing concept, available to all software running under VM, regardless of the version or release level.

Cache Magic automatically creates in-memory cache storage for frequently used DASD files. Because the data is in processor storage, access is instantaneous. I/O activity can be eliminated completely, even between CPU and DASD control unit. As with a cache DASD controller, Cache Magic maintains the most active data in a buffer, avoiding the need to read it from the disk. Requests to write data are reflected immediately to the disk, maintaining data integrity. Because DASD I/O is at CPU speed, it approaches that of solid-state devices, but without the risk of data loss if a power failure occurs or the need to transfer data files from other devices.

Cache Magic is totally transparent, and data is presented to applications exactly as it was previously. Once installed, the full benefits of the product can be obtained without any changes to applications, movement of disk files, or installation of new VM releases.

Manual cacheing of individual cache areas is also permitted, providing users with maximum control of their data. Various configuration options are provided to allow fine tuning. Individual applications or files may be cached as desired.

For further information contact:
SDI, Account 62500, PO Box 210360, Jamaica, NY 11431, USA.
Tel: (650) 572 1200.
SDI UK, PO Box 2360, London, W8 7ZS, UK.
Tel: (0181) 759 8786.
URL: http://www.sdisw.com.

* * *