150

VM update

February 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: xephon@compuserve.com

North American office

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

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.

A concise LISTCAT ALL report

The output generated by a LISTCAT ALL command is a comprehensive source of information; however, for most practical purposes, just a small part of this information is needed. Normally you just want to know how many CI and CA splits a file has, how many extensions or tracks it has allocated, and so on.

Based on this requirement, I have developed a REXX EXEC that reads the output generated by LISTCAT ALL and creates a concise report about the essentials of each cluster.

Figure 1 shows an example of such a report. In the first two columns, there is the cluster name and type (K for KSDS, E for ESDS, R for RRDS, L for Linear, and A for Alternate Index – although not a cluster, I decided to consider it as such since it has data and index components).

The remaining fields (keys, record length, cisize, allocation type, etc) all refer to the data component information.

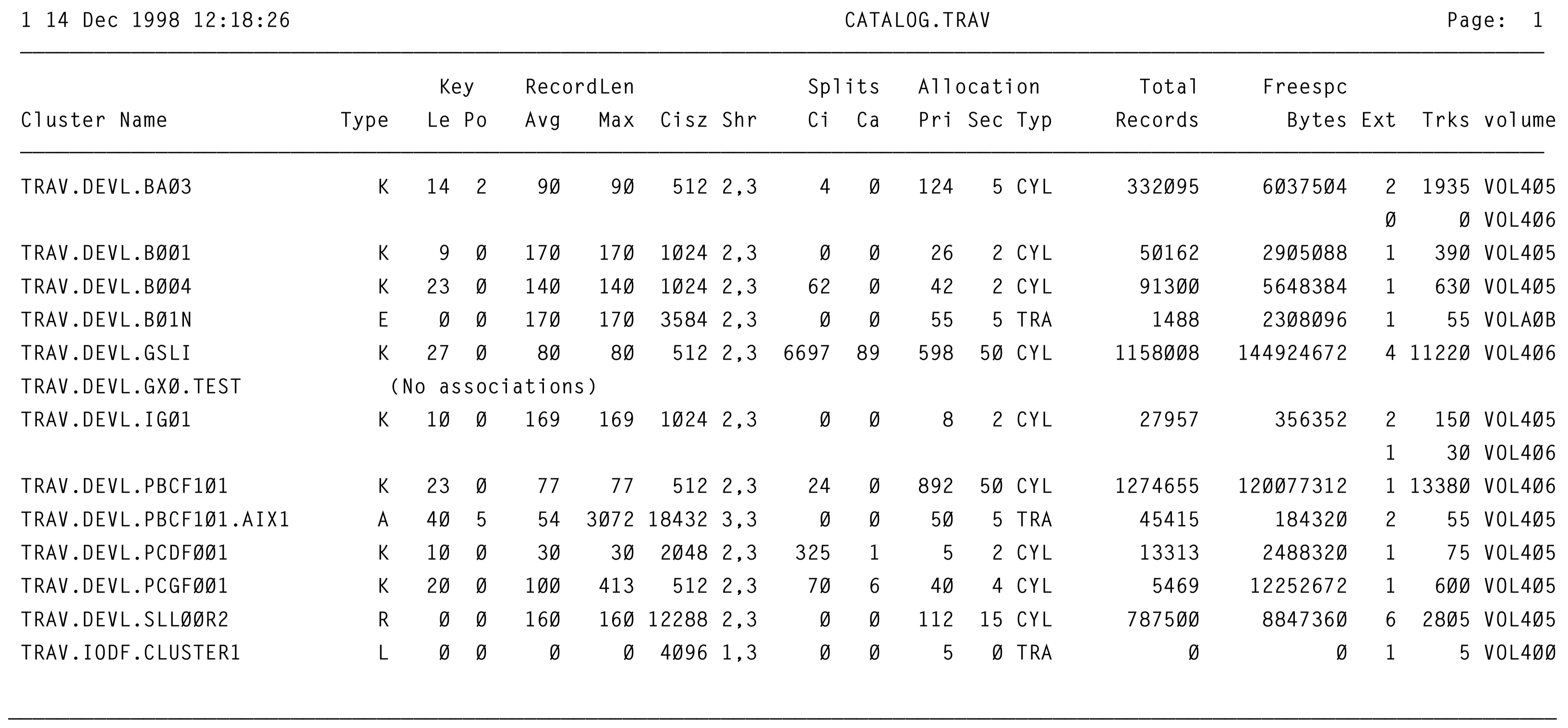
If there is a cluster defined without an associated data element, its line appears with the message '(No associations)'. The last three columns are the number of extents, tracks per volume, and the volume. If a volume is listed as candidate, both tracks and extents will be zero.

RUNNING VSAMLIST UNDER CMS

To run VSAMLIST under CMS, start by creating a CMS file containing the LISTCAT output. After that, at the beginning of the EXEC, add a line setting variable FICIN to the CMS name of that file, or have it passed as an argument, and set variable FICOUT to the CMS name that will hold the report.

For a VM/VSE system, my favourite method to create the LISTCAT output is to link the DASD containing the catalog to be listed, access it with a free drive letter (say 'x'), issue a DLBL IJSYSCT x (VSAM, and run an AMSERV LISTCAT file. This is an 80-byte fixed RECFM CMS file containing just a line with 'LISTCAT ALL CAT(catalog)'.

You might need to use a temporary mini-disk to hold the listing.



```
1 14 Dec 1998 12:18:26                         CATALOG.TRAV                             Page:  1
______________________________________________________________________________________________________

                               Key   RecordLen             Splits   Allocation      Total     Freespc
Cluster Name             Type  Le Po  Avg   Max  Cisz Shr    Ci  Ca  Pri Sec Typ   Records       Bytes Ext  Trks volume
______________________________________________________________________________________________________

TRAV.DEVL.BAØ3              K  14  2   9Ø    9Ø   512 2,3     4   Ø  124   5 CYL    332Ø95     6Ø375Ø4   2  1935 VOL4Ø5
                                                                                                     Ø     Ø VOL4Ø6
TRAV.DEVL.BØØ1              K   9  Ø  17Ø   17Ø  1Ø24 2,3     Ø   Ø   26   2 CYL     5Ø162     29Ø5Ø88   1   39Ø VOL4Ø5
TRAV.DEVL.BØØ4              K  23  Ø  14Ø   14Ø  1Ø24 2,3    62   Ø   42   2 CYL     913ØØ     5648384   1   63Ø VOL4Ø5
TRAV.DEVL.BØ1N              E   Ø  Ø  17Ø   17Ø  3584 2,3     Ø   Ø   55   5 TRA      1488     23Ø8Ø96   1    55 VOLAØB
TRAV.DEVL.GSLI              K  27  Ø   8Ø    8Ø   512 2,3  6697  89  598  5Ø CYL   1158ØØ8   144924672   4 1122Ø VOL4Ø6
TRAV.DEVL.GXØ.TEST           (No associations)
TRAV.DEVL.IGØ1              K  1Ø  Ø  169   169  1Ø24 2,3     Ø   Ø    8   2 CYL     27957      356352   2   15Ø VOL4Ø5
                                                                                                     1    3Ø VOL4Ø6
TRAV.DEVL.PBCF1Ø1           K  23  Ø   77    77   512 2,3    24   Ø  892  5Ø CYL   1274655   12ØØ77312   1 1338Ø VOL4Ø6
TRAV.DEVL.PBCF1Ø1.AIX1      A  4Ø  5   54  3Ø72 18432 3,3     Ø   Ø   5Ø   5 TRA     45415      18432Ø   2    55 VOL4Ø5
TRAV.DEVL.PCDFØØ1           K  1Ø  Ø   3Ø    3Ø  2Ø48 2,3   325   1    5   2 CYL     13313     248832Ø   1    75 VOL4Ø5
TRAV.DEVL.PCGFØØ1           K  2Ø  Ø  1ØØ   413   512 2,3    7Ø   6   4Ø   4 CYL      5469    12252672   1   6ØØ VOL4Ø5
TRAV.DEVL.SLLØØR2           R   Ø  Ø  16Ø   16Ø 12288 2,3     Ø   Ø  112  15 CYL    7875ØØ     884736Ø   6  28Ø5 VOL4Ø5
TRAV.IODF.CLUSTER1          L   Ø  Ø    Ø     Ø  4Ø96 1,3     Ø   Ø    5   Ø TRA         Ø           Ø   1     5 VOL4ØØ

______________________________________________________________________________________________________
```

Figure 1: Example of output

VSAMLIST

```
/*= REXX ===========================================================*/
/*                                                                  */
/*  VSAMLIST: Extracts information from "LISTCAT ALL" listings.     */
/*            The input file for this EXEC is the listing generated */
/*            by LISTCAT ALL CATALOG(catalog) and the output is a   */
/*            file with LRECL=133 and first-column control chars.   */
/*                                                                  */
/*  Running this EXEC                                               */
/*   Under MVS: Allocate DDname FICIN to input and FICOUT to output */
/*   Under VM:  Set variables FICIN and FICOUT to CMS filenames     */
/*                                                                  */
/*==================================================================*/
execio 1 diskr ficin
if rc ¬=0 then do
   say "Error reading input file"
   exit
end
pull linha
cc = left(linha,1)
data_flag = 0
clu = 0
do forever
   execio 1 diskr ficin
   if rc ¬=0 then leave
   pull linha
   if cc then linha=substr(linha,2)
   call select_line_type
end
call write_output
saida:
exit
/*==================================================================*/
/*            Select line type and extract values                   */
/*==================================================================*/
select_line_type:
 select
   when word(linha,1)="LISTING" then do
      catalog = center(word(linha,5),100)
   end
   when substr(linha,1,7)="CLUSTER" |,
        substr(linha,1,3)="AIX" then do
      clu = clu + 1
      cluster.clu = left(word(linha,3),44)
      data.clu = 0
      v = 0
      extent.clu.0 = 0
      data_flag = 0
      if substr(linha,1,3)="AIX" then vstype.clu = "A"
```

```
    end
    when substr(linha,4,4)="DATA" then do
       data.clu = word(linha,3)
       data_flag = 1
    end
    when substr(linha,4,5)="INDEX" then do
       index.clu = word(linha,3)
       data_flag = 0
    end
    when substr(linha,1,7)="NONVSAM" then do
       data_flag = 0
    end
    otherwise nop
  end
  if data_flag then,
  select
    when substr(linha,8,6)="KEYLEN" then do
       linha = translate(linha," ","-")
       keylen.clu = right(word(linha,2),2)
       alrecl.clu = right(word(linha,4),5)
       cisize.clu = right(word(linha,8),5)
    end
    when substr(linha,8,3)="RKP" then do
       linha = translate(linha," ","-")
       keypos.clu = right(word(linha,2),2)
       mlrecl.clu = right(word(linha,4),5)
    end
    when substr(linha,8,8)="SHROPTNS" then do
       shropt.clu = substr(linha,17,3)
       if vstype.clu ¬="A" then do
          type = word(linha,5)
          select
             when type = "NONINDEXED" then vstype.clu = "E"
             when type = "INDEXED"    then vstype.clu = "K"
             when type = "NUMBERED"   then vstype.clu = "R"
             when type = "LINEAR"     then vstype.clu = "L"
             otherwise nop
          end
       end
    end
    when substr(linha,8,7)="REC-TOT" then do
       linha = translate(linha," ","-")
       rectot.clu = right(word(linha,3),11)
       splici.clu = right(word(linha,6),5)
    end
    when substr(linha,8,7)="REC-DEL" then do
       linha = translate(linha," ","-")
       splica.clu = right(word(linha,6),3)
       extent.clu = right(word(linha,8),3)
    end
```

```
   when substr(linha,8,7)="REC-RET" then do
      linha = translate(linha," ","-")
      freeby.clu = right(word(linha,6),11)
   end
   when substr(linha,8,7)="SPACE-T" then do
      linha = translate(linha," ","-")
      sptype.clu = left(word(linha,3),3)
   end
   when substr(linha,8,7)="SPACE-P" then do
      linha = translate(linha," ","-")
      spprim.clu = right(word(linha,3),5)
   end
   when substr(linha,8,7)="SPACE-S" then do
      linha = translate(linha," ","-")
      spseco.clu = right(word(linha,3),4)
   end
   when substr(linha,8,6)="VOLSER" then do
      linha = translate(linha," ","-")
      v = v + 1
      extent.clu.Ø = extent.clu.Ø + 1
      extent.clu.v = word(linha,12)
      volume.clu.v = word(linha,2)
      tracks.clu.v = Ø
   end
   when substr(linha,8,6)="LOW-CC" then do
      linha = translate(linha," ","-")
      tracks.clu.v = tracks.clu.v + word(linha,8)
   end
   otherwise nop
 end
return
/*==================================================================*/
/*                    Write output file                             */
/*==================================================================*/
write_output:
 pagenum = Ø
 lines_per_page = 55
 za="                                       Key   RecordLen     "
 zb="         Splits    Allocation        Total     Freespc"
 zc="Cluster Name                    Type Le Po   Avg   Max  Cisz Shr "
 zd="   Ci  Ca   Pri  Sec Typ     Records       Bytes Ext  Trks Volume"
 z1 = za||zb
 z2 = zc||zd
 zØ = copies("-",131)
 call write_header
 do k = 1 to clu
    line = line + 1
    if line > lines_per_page then call write_header
    if data.k = Ø then do
       queue " "left(cluster.k,34)" (No associations)"
       execio 1 diskw ficout
    end
    else do
       tracks.k = right(tracks.k,5)
       queue " "left(cluster.k,34) vstype.k keylen.k ,
       keypos.k alrecl.k mlrecl.k cisize.k shropt.k ,
       splici.k splica.k spprim.k spseco.k sptype.k ,
       rectot.k freeby.k right(extent.k.1,3) ,
       right(tracks.k.1,5) volume.k.1
       execio 1 diskw ficout
       do j = 2 to extent.k.Ø
          line = line + 1
          queue copies(" ",114) right(extent.k.j,3) ,
          right(tracks.k.j,5) volume.k.j
          execio 1 diskw ficout
       end
    end
 end
 execio Ø diskw ficout "(finis"
return
/*==================================================================*/
/*                  Write output file header                        */
/*==================================================================*/
write_header:
 line = Ø
 pagenum = pagenum+1
 queue "1" date() time() catalog "Page: " pagenum
 queue " "zØ
 queue " "z1
 queue " "z2
 queue " "zØ
 execio 5 diskw ficout
return
```

Luis Paulo Figueiredo Sousa Ribeiro
Systems Programmer
Edinfor (Portugal)

Splitting the XEDIT screen at the cursor position

XEDIT allows you to split the screen in many ways; however, because it is necessary to enter depth, width, start line, and start column for each screen with SET SCREEN DEFINE, most people use equal size splits such as SET SCREEN 3 or SET SCREEN 2 V.

The following macro allows you to split the screen into two, three, or four at the cursor position as detailed in the help file.

CURSPLIT HELPCMS

```
        +----------------------+
        |                      |
        | CURSPLIT XEDIT macro |
        |                      |
        +----------------------+

This macro will split the screen into two, three, or four at the
cursor position.

 CURSPLIT    will split the screen horizontally.
 CURSPLIT V  will split the screen vertically.
 CURSPLIT 4  will split the screen crosswise into four
 CURSPLIT 3  will split the screen into three, one screen full width
         above two others, all screens meeting at the cursor position.

The cursor will then be placed at the start of the command line
in the first logical screen. "SCR 1" will return to one logical
screen.

If the cursor position is such that the screens cannot be defined
correctly then default sizes are used:

   CURSPLIT       - splits horizontally across the middle
   CURSPLIT V     - splits vertically down the middle
   CURSPLIT 3     - the top screen has a third of the full screen depth
                  -  the other screens have half the width
   CURSPLIT 4     - splits halfway down and halfway across

   It is recommended that a PF key should be set to CURSPLIT for
   ease of use. Splitting into 3 or 4 screens will not be
   frequent enough to justify the normal use of a PF key.
```

CURSPLIT XEDIT

```
/***********************************
*  Split screen at cursor position *
***********************************/

'EXTRACT /LSCREEN'   /* get screen dimensions */
                     /* and cursor position   */
parse value cursadd() with physlin physcol .

arg parm .
select
   when parm=''  then call splith   /* horizontally */
   when parm='2' then call splith   /* horizontally */
   when parm='V' then call splitv   /* vertically   */
   when parm='4' then call split4   /* crosswise    */
   when parm='3' then call split3   /* 1 above 2    */
   otherwise 'HELP CURSPLIT'
end
'CURSOR CMDL'
exit

/*********************
* split horizontally *
*********************/
splith:
'SET SCREEN SIZE' physlin lscreen.5-physlin
if rc¬=0 then
do     /* cursor in wrong position */
   'SET SCREEN 2'
   'MSG Split across middle forced'
end
return

/*******************
* split vertically *
*******************/
splitv:
wid1 = physcol             /* across to cursor position */
wid2 = lscreen.6-wid1      /* rest of width of screen   */

'SET SCREEN DEFINE' lscreen.5 wid1 1 1 lscreen.5 wid2 1 wid1+1
if rc¬=0 then
do     /* cursor in wrong position */
   'SET SCREEN 2 V'

   'MSG Split down middle forced'
end
return
```

```
/****************************
* split crossways into four *
****************************/
split4:
wid1 = physcol             /* across to cursor position */
wid2 = lscreen.6-wid1      /* rest of width of screen   */
dep1 = physlin             /* down to cursor position   */
dep2 = lscreen.5-physlin   /* rest of depth of screen   */

do until scrc=Ø
   'SET SCREEN DEFINE' dep1 wid1 1 1 , /* top left     */
      dep1 wid2  1 wid1+1 ,            /* top right    */
      dep2 wid1 dep1+1 1 ,             /* bottom left  */
      dep2 wid2 dep1+1 wid1+1          /* bottom right */
   scrc = rc
   if rc¬=Ø then
   do       /* cursor in wrong position - assume middle of screen */
      dep1 = lscreen.5%2         /* round down half of depth */
      dep2 = lscreen.5 - dep1    /* rest of depth            */
      wid1 = lscreen.6%2         /* round down half of width */
      wid2 = lscreen.6 - wid1    /* rest of width            */
      'MSG Default size forced'
   end
end
return

/*****************************
* split into 3 - one above 2 *
*****************************/
split3:
wid1 = lscreen.6           /* full width for top screen              */
wid2 = physcol             /* across to cursor position for second   */
wid3 = lscreen.6-wid2      /* rest of width of screen for third      */
dep1 = physlin             /* down to cursor position for top screen */
dep2 = lscreen.5-physlin   /* rest of depth of screen for others     */

do until scrc=Ø
   'SET SCREEN DEFINE' dep1 wid1 1 1 , /* across the top */
      dep2 wid2 dep1+1 1 ,             /* bottom left    */
      dep2 wid3 dep1+1 wid2+1          /* bottom right   */
   scrc = rc
   if rc¬=Ø then
   do       /* cursor in wrong position -                  */
            /* assume third of way down and halfway across */
      dep1 = lscreen.5%3         /* round down 1/3 of depth  */
      dep2 = lscreen.5 - dep1    /* rest of depth            */
      wid1 = lscreen.6           /* full width               */
      wid2 = lscreen.6%2         /* round down half of width */
      wid3 = lscreen.6 - wid2    /* rest of width            */
      'MSG Default size forced'
   end
end
return
/***********************************************************
* Return cursor address                                    *
*                                                          *
* Note: The cursor address from EXTRACT/CURSOR cannot be   *
*       be used satisfactorily if the cursor is not in the *
*       same logical screen where the command is entered   *
***********************************************************/

cursadd:
stream = 'Ø3'x           /* read modified command */
'PIPE VAR STREAM',       /* pass value in variable to PIPES        */
'| FULLSCREEN CONDREAD', /* read screen to get cursor address      */
'| 327ØBFRA 2 TO16BIT',  /* convert address from 12-bit to integer */
'| SPECS 2.2 C2D 1',     /* pick out address and make decimal      */
'| VAR CURS'             /* get value into variable                */

   /* physical screen width is in lscreen.6 */
lin = curs%lscreen.6 + 1
col = 1+curs-((lin-1)*lscreen.6)
return lin col
```

John Illingworth
Systems Engineer
Wm Morrison Supermarkets (UK)

A full screen console interface – part 7

Editor's note: this month we continue the code for the full screen console interface for Disconnected Service Machines (DSM). This 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.*

CSCRDF ASSEMBLE

```
         TITLE 'CSCRDF - CSC Read Data File record'
CSCRDF   START X'Ø199EØ'
```

```
         PRINT NOGEN
         CSCHDR                         Read Data file
*
* Read Data File records
*
*
         USING UIDSECT,R8               UID (user) Block
         USING CCHSECT,R7               CCH (cache) Block
         SPACE 3
*
* Return to caller and keep the cc
*
*
RETURN   BACK
         SPACE 3
*
* Read first record from disk (Free List and Cache are not searched)
*
*       Output R7 addresses first record (cache image)
*              If the Data File is empty a non-zero cc is returned
*
*
CSCRDFFT RELOC                          Read first record
         L     R7,CACHEPTR              Address last created record
         L     R4,CCHRECNO              Get record number
RDFF1ØØ  LA    R4,1(,R4)                Increment
         C     R4,DFOLDTOT              After end of physical file
         BNH   RDFF2ØØ
         LA    R4,1                     Yes, go back to first record
         C     R4,DFCURR                Is file empty?
         BH    RETURN                   Yes, record not found
RDFF2ØØ  BAS   R14,GET                  Read record from disk
         TM    DFOPTS-DFBUFF(R1),DFOCONT2 Is it a continuation record?
         BO    RDFF1ØØ                  Yes, skip it and try next one
         IC    R5,DFOPTS-DFBUFF(,R1)    Load option byte
         LA    R1,DFOCYCLE
         NR    R5,R1                    Keep only Cycle bit
         B     RDFN#FT                  Build cache image and test
         SPACE
*
* Locate record by Date and Time
*
*        Input R7 addresses reference record (cache image)
*                 Only CCHDATE and CCHTIME are checked
*       Output R7 addresses requested record (cache image)
*              If the record is not found a non-zero cc is returned
*
*              The first record with Date/Time equal or greater than
*              the specified is returned
*
*
```

```
CSCRDFGO RELOC                         Locate the record
         L     R1,CACHEPTR             Address current record
         L     R1,CCHFWD-CCHSECT(,R1)  Address first cache record
         CLC   CCHDATE,CCHDATE-CCHSECT(R1)
         BL    RDFG3ØØ                 Not there, check Data File
         BH    RDFG1ØØ                 Search cache records
         CLC   CCHTIME,CCHTIME-CCHSECT(R1)
         BNH   RDFG3ØØ                 Not in cache, check Data file
RDFG1ØØ  C     R1,CACHEPTR             All cache searched?
         BE    RDFG9ØØ                 Yes, record not found
         L     R1,CCHFWD-CCHSECT(,R1)  Address next record
         CLC   CCHDATE,CCHDATE-CCHSECT(R1)
         BH    RDFG1ØØ
         BL    RDFG2ØØ                 Record found
         CLC   CCHTIME,CCHTIME-CCHSECT(R1)
         BH    RDFG1ØØ
RDFG2ØØ  LR    R7,R1                   Address record
RDFG21Ø  IC    R5,CCHOPTS              Load option byte
         LA    R1,DFOCYCLE
         NR    R5,R1                   Keep only cycle bit
         B     RDFN#GO                 Now select the right record
         SPACE
RDFG3ØØ  L     R1,CACHEPTR             Record is already on disk
         L     R5,DFOLDTOT             Number of Data File records
         BCTR  R5,Ø                    Do not search last block
         SRL   R5,5                    It could be partially written
         SLL   R5,5                    Each block has 32 DF records
         BNZ   RDFG31Ø
         L     R7,CACHEPTR             Data file is empty
         L     R7,CCHFWD               Use first record from cache
         B     RDFG21Ø
         SPACE
RDFG31Ø  L     R6,CCHRECNO-CCHSECT(,R1) Get current record number
         CR    R6,R5                   If it is the last block or the
         BNH   RDFG32Ø                    Data File is being expanded
         SR    R6,R6                   Do not use any relocation
RDFG32Ø  ST    R6,RDFGRELO             Store relocation value
         SR    R6,R6
RDFG4ØØ  LA    R5,1(,R5)               Add one to number of records
         SRL   R5,1                    Divide by 2, first interval
         AR    R6,R5                   Logical record number to read
         C     R6,DFOLDTOT             If it is after End-Of-File
         BNH   RDFG5ØØ
         L     R6,DFOLDTOT             Read last Data File record
RDFG5ØØ  LR    R4,R6                   Copy to R4
         A     R4,RDFGRELO             Relocate to physical record
         C     R4,DFOLDTOT
         BNH   RDFG51Ø
         S     R4,DFOLDTOT             Wrap around if required
RDFG51Ø  BAS   R14,GET                 Read record from Data File
```

```
         LA    RØ,1                    Is interval down to one?
         CR    RØ,R5
         BE    RDFG6ØØ                 Yes, terminate binary search
         CLC   CCHDATE,DFDATE-DFBUFF(R1) Compare date
         BH    RDFG4ØØ                 Too low, go forward
         BL    RDFG52Ø                 Too high, go backward
         CLC   CCHTIME,DFTIME-DFBUFF(R1)
         BH    RDFG4ØØ
RDFG52Ø  LA    R5,1(,R5)               Add one to interval
         SRL   R5,1                    Divide by two
         SR    R6,R5                   Go backward
         BP    RDFG5ØØ                 If it is before first record
         LA    R6,1                    Read first record
         B     RDFG5ØØ
         SPACE
RDFG6ØØ  TM    DFOPTS-DFBUFF(R1),DFOCONT2 Is this a continuation record
         BZ    RDFG7ØØ                 No, almost done
         LA    R4,1(,R4)               Yes, read next physical record
         C     R4,DFOLDTOT
         BNH   RDFG61Ø
         LA    R4,1                    Wrap around if required
RDFG61Ø  BAS   R14,GET                 Go read the record
         B     RDFG6ØØ                 Check again
         SPACE
RDFG7ØØ  CLC   CCHDATE,DFDATE-DFBUFF(R1) Is this really the record
         BH    RDFG71Ø                 No, it is the following one
         BL    RDFG73Ø                 Yes, we got it
         CLC   CCHTIME,DFTIME-DFBUFF(R1) Maybe...
         BNH   RDFG73Ø
RDFG71Ø  SR    RØ,RØ                   Required by next IC
         IC    RØ,DFCNUM-DFBUFF(,R1)   Number of DF records for message
         AR    R4,RØ                   Address next message
         C     R4,DFOLDTOT
         BNH   RDFG72Ø
         S     R4,DFOLDTOT             Wrap around if required
RDFG72Ø  BAS   R14,GET                 Get next Record
RDFG73Ø  IC    R5,DFOPTS-DFBUFF(,R1)   Load option byte
         LA    R1,DFOCYCLE
         NR    R5,R1                   Keep only cycle bit
         BAS   R14,READREC             Build cache image
         B     RDFN#GO                 Now select the right record
         SPACE
RDFG9ØØ  LTR   R14,R14                 Generate a non-zero cc
         B     RETURN                  Return, record not found
         SPACE
*
* Read next record
*
*        Input R7 points to reference record (cache image)
*        Output R7 addresses next record (cache image)
*              If the record is not found a non-zero cc is returned
*
*              RDFN#FT is invoked by CSCRDFFT
*              RDFN#GO is invoked by CSCRDFGO
*
*
CSCRDFNT RELOC                         Read next record
         IC    R5,CCHOPTS              Load option byte
         LA    R1,CCHCYCLE
         NR    R5,R1                   Keep only Cycle bit
RDFN1ØØ  SR    RØ,RØ                   Required by next IC
         IC    RØ,CCHCNUM              Number of DF records for message
         L     R4,CCHRECNO             Get number of reference record
         AR    R4,RØ                   First record of next message
RDFN#FT  BAS   R14,READREC             Read and build cache image
         BNZ   RETURN                  Not found
RDFN#GO  LINK  SELECT                  Check user selection
         BNZ   RDFN1ØØ                 No good, read next one
         B     RETURN                  We found it
         SPACE
*
* Read last record
*
*       Output R7 addresses last record (cache image)
*              If the record is not found a non-zero cc is returned
*
*
CSCRDFLT RELOC                         Read last record
         LA    R4,1                    Is Data File empty
         C     R4,DFCURR               It is if current record is zero
         BH    RETURN                  File is empty, record not found
         L     R7,CACHEPTR             Address last created record
         LINK  SELECT                  Check user selection
         BZ    RETURN                  Good enough, use it
         B     RDFP#LT                 Try to find it
         SPACE
*
* Read previous record
*
*        Input R7 points to reference record (cache image)
*        Output R7 addresses previous record (cache image)
*              If the record is not found a non-zero cc is returned
*
*              RDFP#LT is invoked by CSCRDFLT
*
*
CSCRDFPR RELOC                         Read previous record
RDFP#LT  IC    R5,CCHOPTS              Load option byte (CSCRDFLT)
         LA    R1,CCHCYCLE
         NR    R5,R1                   Keep only Cycle bit
```

```
RDFP1ØØ  SR    RØ,RØ                   Required by next IC
         IC    RØ,CCHPNUM              DF records for previous message
         L     R4,CCHRECNO             Get number of reference record
         SR    R4,RØ                   First record of previous message
         BP    RDFP9ØØ                 If not positive
         LA    R1,CCHCYCLE
         XR    R5,R1                   Reverse Cycle bit
         A     R4,DFOLDTOT             And wrap around file
RDFP9ØØ  BAS   R14,READREC             Read and build cache image
         BNZ   RETURN                  Not found
         LINK  SELECT                  Check user selection
         BNZ   RDFP1ØØ                 No good, read next one
         B     RETURN                  We found it
         SPACE
*
* Read previous record from disk (Free List and Cache are not searched)
*
*        Input R7 points to reference record (cache image)
*        Output R7 addresses previous record (cache image)
*              If the record is not found a non-zero cc is returned
*
*
CSCRDFDP RELOC                         Read previous record
         IC    R5,CCHOPTS              Load option byte
         LA    R1,CCHCYCLE
         NR    R5,R1                   Get only cycle bit
         SR    RØ,RØ                   Required by next IC
         IC    RØ,CCHPNUM              DF records used by previous msg
         L     R4,CCHRECNO             Current record number
         SR    R4,RØ                   First DF record of previous msg
         BP    RDFD9ØØ                 Read record
         XR    R5,R1                   Swap cycle bit
         A     R4,DFOLDTOT             Wrap around file
RDFD9ØØ  BAS   R14,READDISK            Read and build cache image
         B     RETURN                  Return, cc set by READDISK
         SPACE
*
* Restart Data file
*
*       Output R1 addresses last record written (DF record image)
*              R4 contains the record number pointed by R1
*
*   This routine performs a binary search to locate the require record.
*
*
CSCRDFRS RELOC
         L     R5,DFOLDTOT             Number of record on DF file
         LA    R4,1                    Start with first record
         BAS   R14,GET                 Read the record
         CR    R4,R5                   If DF has ONE record...
         BE    RDFR9ØØ                 We found it
         IC    R6,DFOPTS-DFBUFF(,R1)   Load option byte
         LA    R2,DFOCYCLE
         NR    R6,R2                   Keep only the cycle bit
RDFR1ØØ  LA    R5,1(,R5)               (n + 1) / 2 is the new increment
         SRL   R5,1
         AR    R4,R5                   Go forward
         C     R4,DFOLDTOT             Are we after the last record?
         BNH   RDFR2ØØ
         L     R4,DFOLDTOT             Yes, use the last record
RDFR2ØØ  BAS   R14,GET                 Read the record
         IC    RØ,DFOPTS-DFBUFF(,R1)   Load new option byte
         LA    R2,DFOCYCLE
         NR    RØ,R2                   Get new cycle bit
         LA    R2,1
         CR    R2,R5                   Is the increment down to ONE
         BE    RDFR3ØØ                 Yes, binary search is over
         CR    RØ,R6                   Compare cycle bits
         BE    RDFR1ØØ                 They are the same, go forward
         LA    R5,1(,R5)               They are different...
         SRL   R5,1
         SR    R4,R5                   Go backward...
         B     RDFR2ØØ                 Read next record
         SPACE
RDFR3ØØ  CR    RØ,R6                   Last check, same cycle bits
         BE    RDFR9ØØ                 Yes, we got the record
         BCTR  R4,Ø                    No, use previous record
         BAS   R14,GET                 Read it
RDFR9ØØ  BACK                          All done, return
         SPACE 3
*
*        Input R4 contains the record number to read
*              R5 contains the cycle bit (last byte)
*       Output R7 addresses the record (cache image)
*              If the record is not found a non-zero cc is returned
*
*
READREC  L     R7,UIDFREE2             last record on Free list
         OI    UIDOPT1,UIDFFREE        Set option
READ1ØØ  C     R4,CCHRECNO             Check record number
         BE    READ8ØØ                 Found it...
         L     R7,CCHBWD               Go back one Free entry
         LTR   R7,R7                   Is it the last one
         BNZ   READ1ØØ                 No, test all entries
         NI    UIDOPT1,X'FF'-UIDFFREE  Yes, reset option
         L     R7,CACHEPTR             Try cache buffer
READ2ØØ  C     R4,CCHRECNO
         BE    READ8ØØ                 Found it...
         L     R7,CCHBWD
         C     R7,CACHEPTR             Search all records
         BNE   READ2ØØ
READDISK ST    R14,READSV14            Not found, get it from disk
         C     R4,DFOLDTOT             Out of DF file
         BNH   READ3ØØ
         S     R4,DFOLDTOT             Yes, wrap around
         LA    R1,DFOCYCLE
         XR    R5,R1                   Reverse Cycle bit
READ3ØØ  C     R4,DFOLDTOT             File could be empty...
         BH    READ9ØØ                 Record does not exist
         BAS   R14,GET                 Read the record
         LA    R7,RDFCACHE             Address cache work area
         ST    R4,CCHRECNO             Store record number
         MVC   CCHDFREC,Ø(R1)          Move data from disk record
READ4ØØ  TM    DFOPTS-DFBUFF(R1),DFOCONT1 Multi-record message?
         BZ    READ6ØØ
         LA    R4,1(,R4)               Yes, read next record
         C     R4,DFOLDTOT             End of Data file (physical)
         BNH   READ5ØØ
         S     R4,DFOLDTOT             Wrap around
READ5ØØ  BAS   R14,GET                 Get next record
         SR    R2,R2
         IC    R2,DFRLEN-DFBUFF(,R1)   Get message length (new section)
         SR    R3,R3
         IC    R3,CCHRLEN              Get assembled message length
         LR    RØ,R3
         AR    RØ,R2                   Combine the two parts
         STC   RØ,CCHRLEN              Store new length
         LA    R3,CCHDATA(R3)          Address to move new section
         BCTR  R2,Ø                    Adjust length
         EX    R2,READMVC              Move new part
         B     READ4ØØ                 Build complete message
         SPACE
READ6ØØ  LINK  PREFIX                  Get message prefix
         ST    R5,READSVØ5             Save cycle bit
         LINK  MATCH
         L     R5,READSVØ5             Restore cycle bit
         BNZ   READ7ØØ                 Message not defined
         BAS   R14,CHECK               Check if on Hold
READ7ØØ  L     R14,READSV14            Restore return address
READ8ØØ  SR    RØ,RØ                   Get new cycle bit
         IC    RØ,CCHOPTS
         LA    R1,DFOCYCLE
         NR    RØ,R1
         CR    RØ,R5                   Is it the good one
         BNE   READ9ØØ                 No, record was overwritten
         CR    R14,R14
         BR    R14
         SPACE
READ9ØØ  LTR   R14,R14
         BR    R14
         SPACE
READMVC  MVC   Ø(*-*,R3),DFDATA-DFBUFF(R1)
         SPACE 3
*
* Read a DF record
*
*        Input R4 contains the record number
*        Output R1 addresses the record (DF image)
*
*
GET      EQU   *
         USING RDFSECT,R1
         LR    R3,R4                   Copy record number to read
         BCTR  R3,Ø                    Calculate first record in block
         SRL   R3,5                    That's 32 records / 4K block
         SLL   R3,5
         LA    R3,1(,R3)               We have the record number
         L     R1,RDFPTR               Address first RDF block
GET1ØØ   L     R2,RDFADDR              Address correspondent buffer
         C     R3,RDFREC               Check record number
         BE    GET2ØØ                  We found the buffer
         L     R1,RDFFWD               Check next buffer
         C     R1,RDFPTR               Is it the last buffer
         BNE   GET1ØØ
         L     R1,RDFFWD               Yes, we need to read it
         ST    R1,RDFPTR               Select next RDF block
         L     R2,RDFADDR              Address buffer
         ST    R3,RDFREC               Store number of first record
         FSREAD FSCB=DFFILER,FORM=E,BUFFER=(R2),RECNO=(R3)
         LTR   R15,R15
         BZ    GET2ØØ                  We did it
         MSG   Ø17Ø,RC                 Read error, close the shop
         LINK  CLOSE
         SPACE
GET2ØØ   LR    R1,R4                   Copy record number to read
         SR    R1,R3                   Calculate record offset
         SLL   R1,7                    DF record is 128 bytes long
         LA    R1,Ø(R1,R2)             Address required record
         BR    R14
         DROP  R1
         SPACE 3
*
* Check messages on Hold
*
*
CHECK    EQU   *
         TM    CCHOPTS,CCHHOLD         Is message on Hold
         BZR   R14                     No, all done
         L     R1,HLDPTR               Get list of messages
CHECK1ØØ LTR   R1,R1                   Do we have one
```

```
         BZ    CHECK9ØØ                No, reset option
         CLC   CCHRECNO,CCHRECNO-CCHSECT(R1) Check record number
         BNE   CHECK8ØØ
         CLC   CCHDATE,CCHDATE-CCHSECT(R1)         name
         BNE   CHECK8ØØ
         CLC   CCHTIME,CCHTIME-CCHSECT(R1)         time
         BNE   CHECK8ØØ
         CLC   CCHUSER,CCHUSER-CCHSECT(R1)         user-id
         BNE   CHECK8ØØ
         BR    R14                     Found, still not released
         SPACE
CHECK8ØØ L     R1,CCHFWD-CCHSECT(,R1)  Scan all list
         B     CHECK1ØØ
         SPACE
CHECK9ØØ NI    CCHOPTS,X'FF'-CCHHOLD   Message already released, reset
         ST    R14,CHECSV14
         LINK  PREFIX                  Restore also attributes
         L     R14,CHECSV14
         BR    R14
         SPACE 3
         DS    ØD
RDFCACHE DS    CL256                   Area to build cache image
READSV14 DS    F                       Save R14     READDISK
CHECSV14 DS    F                                    CHECK
READSVØ5 DS    F                       Save R5      READDISK
RDFGRELO DS    F                       Relocating record for CSCRDFGO
         SPACE
         CSCDATA
         CSCDS (UID,CCH,RDF)
         REGEQU
         END
```

CSCCPW ASSEMBLE

```
         TITLE 'CSCCPW - CSC Write CP message on disk'
CSCCPW   START X'Ø15668'
         PRINT NOGEN
         CSCHDR                        Write disk file
*
* Write CP message on disk
*
*
         USING IPARML,R9               IUCV Parameter List
         USING UIDSECT,R8              UID (user) Block
         USING CCHSECT,R7              CCH (cache) Block
         BAS   R14,CACHEREC            Move record into cache
         LINK  PREFIX                  Move record prefix
         LINK  MATCH                   Check message
         ST    R5,CPWRSVØ5             Save MSG entry address or zero
         SR    R1,R1                   Required by next IC
         IC    R1,CCHRLEN              Get message length
         LA    R2,CCHDATA(R1)          Address end of message
         LA    R6,CCHDATA              Address message
         MVC   DFBUFF(DFDATA-DFBUFF),CCHDFREC Move date, time, etc...
CPWR100  LA    R0,L'DFDATA             Length of data area
         LR    R1,R2                   Last byte of message
         SR    R1,R6                   Length of message
         CR    R1,R0
         BNH   CPWR200
         LR    R1,R0                   Too big, split
CPWR200  STC   R1,DFRLEN               Store data length
         LTR   R1,R1                   Is length zero
         BNP   CPWR210                 Yes, no need to move data
         BCTR  R1,0                    Prepare to EXecute
         EX    R1,CPWRMVC              Move data
         LA    R6,1(R1,R6)             Update pointer
CPWR210  L     R0,DFCURR               Last data record written
         C     R0,DFOLDTOT             Actual last record on file
         BL    CPWR600
         BE    CPWR500
         L     R1,DFEXPLIN             We are expanding
         LA    R1,1(,R1)               Number of expanded records
         ST    R1,DFEXPLIN
         LR    R1,R0                   Last record written
         SRL   R1,5                    Is record number multiple of 32?
         SLL   R1,5
         CR    R1,R0                   Is block full? (4K = 32 * 128)
         BNE   CPWR300
         ST    R0,DFOLDTOT             Yes, commit expansion
         FSCLOSE FSCB=DFFILEW
         FSOPEN FSCB=DFFILEW,FORM=E,CACHE=NO,OPENTYP=WRITE
CPWR300  L     R0,DFCURR               Last record written
         C     R0,DFNEWTOT             New data file size
         BL    CPWR800
         C     R0,DFOLDTOT             Expansion completed
         BE    CPWR400                 Commit if necessary
         ST    R0,DFOLDTOT
         FSCLOSE FSCB=DFFILEW
         FSOPEN FSCB=DFFILEW,FORM=E,CACHE=NO,OPENTYP=WRITE
CPWR400  MSG   0160                    Display expansion completed msg
         B     CPWR700                 Process record
         SPACE
CPWR500  C     R0,DFNEWTOT             Check against new file size
         BNL   CPWR700
         MSG   0161                    Begin Data file expansion
         B     CPWR800
         SPACE
CPWR600  C     R0,DFNEWTOT             Check new Data file size
         BNE   CPWR800
         ST    R0,DFOLDTOT              Store new Data file size
         LA    R1,1                     Prepare to truncate file
         AR    R0,R1
         LA    R1,DFFILEW
         USING NUCON,R0
         USING FSCBD,R1
         ST    R0,FSCBAITN              Store new limit into FSCB
         DMSKEY NUCLEUS                 Get CMS nucleus key
         L     R15,ATRUNC               Truncate file
         DROP  R0,R1
         BASR  R14,R15
         DMSKEY RESET                   Reset storage key
         MSG   0162                     Display file truncated message
         B     CPWR700
         SPACE
CPWR700  SR    R0,R0                    Go back to the begin
         XI    DFOPTS,DFOCYCLE          Swap cycle bit
         TM    DFOPTS,DFOCONT2          Is it first or only record?
         BO    CPWR800
         XI    CCHOPTS,DFOCYCLE         Yes, also update cache record
CPWR800  LA    R1,1                     Increment record pointer
         AR    R0,R1
         ST    R0,DFCURR                Store it
         TM    DFOPTS,DFOCONT2          Is it first or only record?
         BO    CPWR810
         ST    R0,CCHRECNO              Yes, store record number (cache)
CPWR810  A     R1,DFSSSLIN              Increment number of messages
         ST    R1,DFSSSLIN                 processed during this session
         CR    R6,R2                    Is message complete
         BE    CPWR820
         OI    DFOPTS,DFOCONT1          No, set continuation bit
CPWR820  LR    R1,R0                    Record number to be written
         SRL   R1,5                     Calculate number of last record
         SLL   R1,5                     ...in the block (32 records)
         CR    R1,R0                    Is it last record of block
         BNE   CPWR850                  No, keep going
         SRL   R1,5                     Yes, get first record in block
         BCTR  R1,0
         SLL   R1,5
         LA    R0,1(,R1)                First record of current block
         L     R1,RDFPTR                Address first read buffer
         USING RDFSECT,R1
CPWR830  L     R1,RDFFWD                Check all buffers
         C     R0,RDFREC                Compare record numbers
         BE    CPWR840
         C     R1,RDFPTR                Process all buffers
         BNE   CPWR830
         B     CPWR850
         SPACE
CPWR840  XC    RDFREC,RDFREC            We found it, invalidate buffer
```

```
         DROP  R1
         SPACE
CPWR85Ø  L     RØ,DFCURR                Record number to write
         FSWRITE FSCB=DFFILEW,FORM=E,RECNO=(RØ)
         LTR   R15,R15
         BZ    CPWR86Ø
         MSG   Ø163,RC                  We got a problem, close the shop
         LINK  CLOSE
         SPACE
CPWR86Ø  TM    DFOPTS,DFOCONT1          Is message to be continued?
         BZ    CPWR9ØØ                  No, done
         NI    DFOPTS,X'FF'-DFOCONT1    Yes, reset continuation bit
         OI    DFOPTS,DFOCONT2          Set continued bit
         B     CPWR1ØØ                  Loop back
         SPACE
CPWR9ØØ  BAS   R14,BRDCAST              Broadcast message
         BACK
         SPACE
CPWRMVC  MVC   DFDATA(*-*),Ø(R6)        Move data into DFFILE record
         SPACE 3
*
* Move record into cache
*
*
CACHEREC EQU   *                        Move record into cache
         ST    R14,CACHSV14
         LA    R6,CSCBUFF               Address message
         LA    R1,DIAGØØØC              Work area for DIAG
         DIAG  R1,RØ,X'ØØØC'            Get date and time
         L     R7,CACHEPTR              Last entry updated
         IC    RØ,CCHCNUM               Records on Data File
         L     R7,CCHFWD                Address next entry
         STC   RØ,CCHPNUM               Records on DF for previous cache
         MVC   CCHDATE(2),DIAGØØØC+6    Edit date to yy/mm/dd format
         MVI   CCHDATE+2,C'/'
         MVC   CCHDATE+3(5),DIAGØØØC
         MVC   CCHTIME,DIAGØØØC+8       Move time
         MVC   CCHUSER,Ø(R6)            Move origin user-id from message
         MVC   CCHOPTS,DFOPTS           Reset all options but cycle bit
         NI    CCHOPTS,DFOCYCLE
         LA    R6,8(,R6)                Skip *MSG user-id
         LA    RØ,CLSCIF
         C     RØ,IPTRGCLS
         BNE   CACH1ØØ
         MVC   CCHUSER,Ø(R6)            Use user-id from SCIF instead
         LA    R6,1Ø(,R6)               Skip SCIF user-id
CACH1ØØ  CLI   2(R6),C':'               Check for time stamp
         BNE   CACH12Ø
         CLI   5(R6),C':'
         BNE   CACH12Ø
         LA    RØ,8(,R6)                Is it from current message?
         C     RØ,CSCBUFFE
         BH    CACH2ØØ                 No, left over from previous one
*        MVC   CCHTIME,Ø(R6)           Move time to record prefix
         LA    R6,8(,R6)
         CLI   Ø(R6),C' '
         BNE   CACH12Ø
         LA    R6,1(,R6)
CACH12Ø  CLC   CCHUSER,Ø(R6)           Skip user-id from message
         BNE   CACH2ØØ
         LA    R6,8(,R6)
         CLI   Ø(R6),C' '
         BNE   CACH2ØØ
         LA    R6,1(,R6)
CACH2ØØ  LA    RØ,L'CCHDATA            Length of data area
         L     R1,CSCBUFFE             End address of message
         SR    R1,R6                   Length of message
         CR    R1,RØ
         BNH   CACH21Ø
         LR    R1,RØ                   Too big, truncate
CACH21Ø  STC   R1,CCHRLEN              Store data length
         LA    RØ,1                    Find out how many DF records...
         C     R1,DFLR1                ... are required for this cache
         BNH   CACH22Ø
         LA    RØ,2
         C     R1,DFLR2
         BNH   CACH22Ø
         LA    RØ,3
CACH22Ø  STC   RØ,CCHCNUM
         LTR   R1,R1                   Is length zero
         BNP   CACH23Ø                 Yes, no need to move data
         BCTR  R1,Ø                    Prepare to EXecute
         EX    R1,CACHMVC              Move data
CACH23Ø  ST    R7,CACHEPTR             Save pointer to current entry
         L     R14,CACHSV14
         BR    R14
         SPACE
CLSCIF   EQU   8                       SCIF message class for *MSG
CACHMVC  MVC   CCHDATA(*-*),Ø(R6)      Move data into cache record
         SPACE 3
*
* Broadcast
*
*
         USING MSGSECT,R5
BRDCAST  EQU   *                       Broadcast
         ST    R14,BRDCSV14
         LTR   R5,R5                   Check MATCH result
         BZ    BRDC2ØØ                 No special processing
         TM    MSGOPTS,MSGORTE         Is message to be routed?
         BZ    BRDC1ØØ
         BAS   R14,ROUTE               Yes, do it
```

```
BRDC100  TM    MSGOPTS,MSGORLS         Does message release others?
         BZ    BRDC110
         BAS   R14,RELEASE             Check messages to release
BRDC110  TM    MSGOPTS,MSGUNIQ         Is message to be held unique?
         BZ    BRDC120
         BAS   R14,UNIQUE              Release previous messages
BRDC120  TM    MSGOPTS,MSGHOLD         Is message to be held?
         BZ    BRDC130
         BAS   R14,HOLD                Add message to HOLD list
BRDC130  TM    MSGOPTS,MSGOEXT         Exit EXEC requested?
         BZ    BRDC190
         BAS   R14,EXIT                Invoke Exit EXEC
BRDC190  TM    MSGOPTS,MSGNODSP        NoDisplay message?
         BO    BRDC800                 Yes, almost done...
BRDC200  LA    R8,SSSPTR               Address list of active sessions
         SPACE
BRDC300  L     R8,UIDFWD               Address active session
         LTR   R8,R8
         BZ    BRDC800                 All checked, refresh screens
         TM    UIDOPT2,UIDAUTO         Is session in auto refresh?
         BO    BRDC310                 Yes, check message
         TM    UIDOPT3,UIDCMS          Is CMS scroll active
         BZ    BRDC300
         TM    UIDOPT3,UIDCLEAR        Yes, was screen cleared before
         BZ    BRDC300
         NI    UIDOPT3,X'FF'-UIDCLEAR  Yes, reset clear option
         L     R0,CCHRECNO             Load current record number
         ST    R0,UIDCMSTP             Store as new top line
         B     BRDC300
         SPACE
BRDC310  L     R7,CACHEPTR             Address current record
         LINK  SELECT                  Is message expected by the user?
         BNZ   BRDC300                 No, check another one
         TM    UIDOPT1,UIDRLSE         Any message released already
         BO    BRDC300                 Wait, we must rebuild the screen
         L     R7,UIDBUFF1             Start with first msg on screen
BRDC400  TM    CCHOPTS,CCHHOLD         Is it on Hold
         BZ    BRDC500
         C     R7,UIDBUFF2             Yes, is it the last detail line?
         BE    BRDC300                    Yes, check other sessions
         L     R7,CCHFWD               Try next screen line
         B     BRDC400
         SPACE
BRDC500  TM    UIDOPT3,UIDCMS          CMS scrolling?
         BO    BRDC600                 Yes, process CMS style
         LINK  DELETE                  Delete first scrollable line
         L     R1,UIDBUFF2             Address last line on screen
         B     BRDC700                 Add line and refresh user screen
         SPACE
BRDC600  TM    UIDOPT3,UIDCLEAR        Was screen cleared?
         BO    BRDC630                 Yes, so clear it again
```

```
BRDC61Ø  L     R1,CCHRECNO             Is line in use?
         LTR   R1,R1
         BNZ   BRDC62Ø                 Yes, try next one
         CLI   CCHUSER,X'ØØ'           Is it a blank line?
         BNE   BRDC62Ø                 No, keep trying
         TM    UIDOPT3,UIDWRAP         Yes, is Message Wrap active?
         BZ    BRDC65Ø                    No, use the line
         CLI   CCHLINE2,X'ØØ'          Is line displayable?
         BE    BRDC63Ø                 No, try clear the screen
         B     BRDC65Ø                 Yes, use it
         SPACE
BRDC62Ø  L     R7,CCHFWD               Address next line
         LTR   R7,R7
         BNE   BRDC61Ø                 Check all lines
BRDC63Ø  LINK  CLEAR                   Screen full, clear scroll lines
         L     R7,CACHEPTR             Address current record
         L     RØ,CCHRECNO             Load record number
         ST    RØ,UIDCMSTP             Save as new CMS top line
         NI    UIDOPT3,X'FF'-UIDCLEAR  Reset Clear option
         L     R7,UIDBUFF1             Start with first msg on screen
BRDC64Ø  TM    CCHOPTS,CCHHOLD         Is it on Hold
         BZ    BRDC65Ø                 No, delete and add new one
         L     R7,CCHFWD               Yes, skip it
         B     BRDC64Ø                 Locate message to replace
         SPACE
BRDC65Ø  L     R4,CCHBWD               Address previous line
         LINK  DELETE                  Delete first free line
         LR    R1,R4                   Add after previous...
BRDC7ØØ  L     R7,CACHEPTR             Address current line
         LINK  ADD                     Add current line
         OI    UIDOPT4,UIDBSCR         Option to rebuild user screen
         L     R5,CPWRSVØ5             Restore MSG entry address
         LTR   R5,R5                   Entry found for this message?
         BZ    BRDC71Ø                 No, keep going
         TM    MSGOPTS,MSGALARM        Should we beep beep?
         BZ    BRDC71Ø
         OI    UIDOPT4,UIDBALM         Yes, set Alarm option
BRDC71Ø  TM    UIDOPT3,UIDWRAP         Is Message Wrap active?
         BZ    BRDC72Ø
         GO    CSCWRP                  Yes, build partial lines
BRDC72Ø  TM    UIDOPT1,UIDCONN         Is user connected?
         BO    BRDC3ØØ                 Yes, there is no need do it
         TM    UIDOPT1,UIDRMTE         Is user remote?
         BO    BRDC73Ø                 Yes, send data back
         GO    CSCBLD                  Rebuild user screen (327Ø DS)
         LINK  SEND                    Send it
         B     BRDC3ØØ
         SPACE
BRDC73Ø  GO    CSCUSADP                Send data back to user
         B     BRDC3ØØ
         SPACE
BRDC8ØØ  L     R5,CPWRSVØ5             Restore MSG entry address
         LTR   R5,R5                   Entry found for this message?
         BZ    BRDC9ØØ                 No, that's all
         TM    MSGOPTS,MSGORLS+MSGUNIQ Was message releasing messages?
         BZ    BRDC9ØØ
         GO    CSCURLRF                Yes, refresh rlsd msgs screens
BRDC9ØØ  L     R14,BRDCSV14            Return
         BR    R14
         SPACE 3
*
* Release messages (Name / Release option)
*
*
RELEASE  EQU   *                       Release messages
         ST    R14,RELESV14
         LA    R2,MSGRLSE              Address Release name
         L     RØ,MSGPTR               Address MSG Table
RELE1ØØ  LTR   R5,RØ                   End of MSG Table?
         BZ    RELE9ØØ                 Yes, all done
         L     RØ,MSGFWD               Address next entry
         CLC   MSGNAME,Ø(R2)           Compare Name with Release
         BNE   RELE1ØØ                 Not this one
         L     R1,HLDPTR               Found it now scan the Hold Table
RELE2ØØ  LTR   R7,R1                   End of table?
         BZ    RELE1ØØ                 Yes, check all MSG entries
         L     R1,CCHFWD               Address next message
         C     R5,CCHBWD               Check MSG address that cause Hol
         BNE   RELE2ØØ
         STM   RØ,R3,RELESAVE          Found it, save work registers
         GO    CSCURLPR                Release message
         LM    RØ,R3,RELESAVE          Restore work registers
         B     RELE2ØØ                 Check all messages
         SPACE
RELE9ØØ  L     R7,CACHEPTR             Restore pointer to current line
         L     R5,CPWRSVØ5             Restore MSG entry address
         L     R14,RELESV14
         BR    R14
         SPACE 3
*
* Process Unique messages
*
*
UNIQUE   EQU   *                       Process Unique messages
         ST    R14,UNIQSV14
         L     R1,HLDPTR               Address messages on Hold
UNIQ1ØØ  LTR   R7,R1                   Any message left?
         BZ    UNIQ9ØØ                 No, all done
         L     R1,CCHFWD               Address next message
         C     R5,CCHBWD               Check Hold MSG entry
         BNE   UNIQ1ØØ                 Not this one
         GO    CSCURLPR                Release message
UNIQ9ØØ  L     R7,CACHEPTR             Address current line
         L     R14,UNIQSV14
         BR    R14
         SPACE 3
*
*  Add message to Hold list
*
*  Note: Backward pointer CCHBWD is used to save the MSGSECT address
*        of the rule that put this message on Hold.
*        Used to release UNIQUE messages.
*
*
HOLD     EQU   *                       Hold message
         ST    R14,HOLDSV14
         LA    RØ,CCHSIZE
         LINK  OBTAIN                  Allocate storage
         MVC   Ø(CCHSIZEB,R1),CCHSECT  Copy message
         L     R2,HLDLAST              Address last entry
         LTR   R2,R2                   Is this the first message?
         BNZ   HOLD1ØØ
         ST    R1,HLDPTR               Yes, store table address
         B     HOLD9ØØ
         SPACE
HOLD1ØØ  ST    R1,CCHFWD-CCHSECT(,R2)  Chain with old last message
HOLD9ØØ  SR    RØ,RØ
         ST    RØ,CCHFWD-CCHSECT(,R1)  Clear forward pointer
         ST    R5,CCHBWD-CCHSECT(,R1)  Save MSGSECT address
         ST    R1,HLDLAST              This is the new last message
         L     R14,HOLDSV14
         BR    14
         SPACE 3
*
* Invoke Exit EXEC
*
*
EXIT     EQU   *                       Invoke Exit EXEC
         USING FSCBD,R1
         ST    R14,EXITSV14
         LA    R1,EXFILE               Address FSCB
         MVC   FSCBFN,MSGEXIT          Move Exit name into FSCB
         FSSTATE FSCB=EXFILE           Verify if EXEC exists
         LTR   R15,R15                 Yes, invoke exit EXEC
         BZ    EXIT1ØØ
         LA    R2,MSGEXIT              No, address exit name
         MSG   Ø164                    Display error message
         B     EXIT9ØØ
         SPACE
EXIT1ØØ  MVC   EXPLFN,MSGEXIT          Move name into Parameter List
```

```
         MVC   EXEPLMSG,MSGEXIT         Build also EPL
         LA    R1,EXEPLMSG+L'MSGEXIT    Address end of EXEC name
EXIT2ØØ  BCTR  R1,Ø                     Remove traling blanks
         CLI   Ø(R1),C' '
         BE    EXIT2ØØ
         MVI   1(R1),C' '               Make sure we have one blank
         LA    R1,2(,R1)                Address to move message
         SR    R2,R2                    Required by next IC
         IC    R2,CCHRLEN               Load message length
         LA    R2,CCHDATA-CCHDFREC(,R2) Add DF prefix length
         LA    RØ,Ø(R2,R1)              Calculate end address of message
         ST    RØ,EXEPLEND              Store into Extended PL
         BCTR  R2,Ø                     Prepare to Execute
         EX    R2,EXMVC                 Move DF record into EPL
         TM    CSCFLGØ1,HNDIOS          Check for Console trap
         BZ    EXIT3ØØ
         HNDIO CLR,DEVNAME=CONS         Disable trap
EXIT3ØØ  CMSCALL PLIST=EXPL,EPLIST=EXEPL,COPY=NO Invoke exit EXEC
         TM    CSCFLGØ1,HNDIOS
         BZ    EXIT9ØØ
         WAITT                          Wait for I/O to complete
         L     R2,ADDRCONS
         L     R3,@CSCIOX
         LA    R4,IOXBK
         HNDIO SET,DEVNAME=CONS,DEVICE=(R2),EXIT=(R3),                 *
               INTBLOK=((R4),L'IOXBK)
EXIT9ØØ  L     R14,EXITSV14
         BR    R14
         SPACE
EXMVC    MVC   Ø(*-*,R1),CCHDFREC       Move DF record into EPL
         DROP  R1
         SPACE 3
*
*  Route a message to one or more users
*
*
*
ROUTE    EQU   *                        Route message
         USING RTESECT,R3
         ST    R14,ROUTSV14
         L     RØ,RTEPTR                Address Route table
         SR    R4,R4                    Zero counter
ROUT1ØØ  LTR   R3,RØ                    Check for End of table
         BZ    ROUT6ØØ
         L     RØ,RTEFWD                Not yet, address following entry
         CLC   MSGROUTE,RTENAME         Compare route name
         BNE   ROUT1ØØ                  Not this one, try next
         SR    R6,R6                    Route entry found
         IC    R6,RTECNT                Load number of Node/User pairs
ROUT2ØØ  LR    R1,R6                    Copy
```

```
         BCTR  R1,Ø                    Calculate offset
         SLL   R1,4                    That's 16 bytes per pair
         LA    R1,RTENODE(R1)          Address correct Node/User
         CLC   CSCNODE,Ø(R1)           Check node
         BE    ROUT3ØØ                 It is the same, use CP to send
         BAS   R14,SENDRSCS            Not the same, use RSCS
         B     ROUT4ØØ
         SPACE
ROUT3ØØ  LA    R1,L'RTENODE(,R1)       Address destination user
         BAS   R14,SENDCP              Build and send message
ROUT4ØØ  LA    R4,1(,R4)               Count messges sent
         BCT   R6,ROUT2ØØ              Process all Node/User pairs
         L     RØ,RTEFWD               Process all Route table
         B     ROUT1ØØ
         SPACE
ROUT6ØØ  LTR   R4,R4                   Did we send any message?
         BNZ   ROUT9ØØ                 Yes, all done
         LA    R1,MSGROUTE             No, use route name as user-id
         BAS   R14,SENDCP              Send message to the same node
ROUT9ØØ  L     R14,ROUTSV14
         BR    R14
         SPACE
*
* Build message (RSCS)
*
*        Input R1 points to NODE/USER entry
*
*
SENDRSCS EQU   *
         LA    R2,CPWTEXT              Address message work area
         MVC   Ø(L'CPWSMSG,R2),CPWSMSG Move RSCS communication command
         MVI   L'CPWSMSG(R2),C' '      Force a blank separator
         LA    R2,L'CPWSMSG+1(,R2)
         MVC   Ø(L'CSCRSCS,R2),CSCRSCS Move RSCS user-id
         MVI   L'CSCRSCS(R2),C' '
         LA    R2,L'CSCRSCS+1(,R2)
         MVC   Ø(L'CPWMSG,R2),CPWMSG   Move RSCS MSG command
         LA    R2,L'CPWMSG(,R2)
         MVC   Ø(L'RTENODE,R2),Ø(R1)   Move destination Node-id
         MVI   L'RTENODE(R2),C' '
         LA    R2,L'RTENODE+1(,R2)     Next free byte in message area
         LA    R1,L'RTENODE(,R1)       Address destination user-id
         B     SENDALL                 EXECute CP/RSCS common code
         SPACE
*
* Build message (CP)
*
*        Input R1 points to USER
*
*
```

```
SENDCP   EQU   *
         LA    R2,CPWTEXT              Address message area
         MVC   Ø(L'CSCMSGC,R2),CSCMSGC Move CP command (MSG or MSGNOH)
         MVI   L'CSCMSGC(R2),C' '      At least one space is required
         LA    R2,L'CSCMSGC+1(,R2)     Advance pointer
         SPACE
SENDALL  EQU   *                       Common code to CP and RSCS
         MVC   Ø(8,R2),Ø(R1)           Move destination user-id
         LA    R2,8(,R2)               Skip user-id
         MVI   Ø(R2),C' '              Force a blank separator
SEND1ØØ  BCTR  R2,Ø                    Check for multiple blanks
         CLI   Ø(R2),C' '
         BE    SEND1ØØ                 Found one, remove it
         MVC   2(L'CPWMSGB,R2),CPWMSGB Move message header
         LA    R2,L'CPWMSGB+2(,R2)
         MVC   Ø(L'CCHUSER,R2),CCHUSER Move originating user-id
         LA    R2,L'CCHUSER(,R2)
SEND2ØØ  BCTR  R2,Ø                    Remove all blanks
         CLI   Ø(R2),C' '
         BE    SEND2ØØ
         MVC   1(L'CPWMSGE,R2),CPWMSGE Close message header (:)
         LA    R2,L'CPWMSGE+1(,R2)
         LA    RØ,CPWTEXT+L'CPWTEXT    Address end of message area
         SR    RØ,R2                   Calculate amount of free space
         SR    R1,R1
         IC    R1,CCHRLEN              Load message length
         CR    RØ,R1                   Space enough?
         BNL   SEND3ØØ
         LR    R1,RØ                   No, truncate message
SEND3ØØ  BCTR  R1,Ø                    Prepare to Execute
         EX    R1,SENDMVC              Move message text
         LA    R2,1(R1,R2)             Address end of message
         LA    RØ,CPWTEXT              Address message area
         SR    R2,RØ                   Calculate message length
         O     R2,CPWRESP              Request CP response in buffer
         LA    R1,CSCBUFF              Address response buffer
         ST    R3,SENDSVØ3             Save R3
         LA    R3,1                    Buffer length (dummy)
         DIAG  RØ,R2,X'ØØØ8'           Call CP to EXECute command
         L     R3,SENDSVØ3             Restore R3
         BR    R14
         SPACE
SENDMVC  MVC   Ø(*-*,R2),CCHDATA       Move message text
         SPACE
         DROP  R3,R5
         SPACE 3
CACHSV14 DS    F                       Save R14   CACHEREC
BRDCSV14 DS    F                                  BRDCAST
RELESV14 DS    F                                  RELEASE
UNIQSV14 DS    F                                  UNIQUE
```

```
HOLDSV14 DS    F                                  HOLD
EXITSV14 DS    F                                  EXIT
ROUTSV14 DS    F                                  ROUTE
SENDSVØ3 DS    F                            R3    SEND
CPWRSVØ5 DS    F                            R5    CPW (MSG entry addr)
RELESAVE DS    4F                           RØ-R3 RELEASE
         SPACE
@SCURLPR DC    V(CSCURLPR)             Release messages
@SCURLRF DC    V(CSCURLRF)             Refresh released messages scrns
DFLR1    DC    A(L'DFDATA)             Maximum length for 1 DF record
DFLR2    DC    A(L'DFDATA*2)           Maximum length for 2 DF records
         SPACE
CPWTEXT  DS    CL128                   Area to build CP/RSCS message
         SPACE
CPWRESP  DC    X'4ØØØØØØØ'             Request CP response in buffer
CPWSMSG  DC    C'SMSG '                RSCS communication command
CPWMSG   DC    C' MSG '                RSCS MSG command
CPWMSGB  DC    C'<CSC> '               Message header
CPWMSGE  DC    C': '                   Termination of message header
         SPACE
         DS    ØD
EXPL     DC    C'EXEC    '             Parameter List for Exit EXEC
EXPLFN   DC    C'        '
         DC    X'FFFFFFFFFFFFFFFF'
EXEPLMSG DS    CL256                   Message that invoked exit
EXEPL    DC    A(EXPL)             *1* Extended Parameter List
         DC    A(EXEPLMSG)         *2*
EXEPLEND DC    A(*-*)              *3*
         DC    A(Ø)                *4* Extended Parameter List word 4
EXFILE   FSCB  '* EXEC *',FORM=E
         SPACE 3
         CSCDATA
         CSCDS (CCH,UID,RDF,MSG,RTE)
         NUCON
         FSCBD
         REGEQU
         PRINT OFF
         COPY  IPARML
         PRINT ON
         END
```

It is now possible to generate CSCSVP. The module will collect the data and create the log file, but you cannot establish user sessions yet. This will be possible after adding CSCSCN, CSCBLD, CSCUSC, CSCUIN, and CSCSEV.

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

Fernando Duarte
Analyst (Canada)

Mouse on the mainframe

In this second article on the manipulation of System/390 applications with a PC or workstation mouse, the author discusses writing REXX programs with virtual screens and CMS windows.

INTRODUCTION

In a previous article in *VM Update*, Issue 146, October 1998, I discussed the concept and rationale for writing user-oriented System/390 applications that can be manipulated with a PC or workstation mouse. 'Pointer Enabled Tools' or PETs were proposed as productivity enhancements because clicking with a mouse on predefined screen 'hot spots' takes considerably less effort and is less error-prone than using keystrokes. Novice or casual VM/CMS users find the PETs style interface dramatically easier to master than the standard command line interface.

This article outlines one way in which PETs applications can be written using REXX, CMS virtual screens and windows, and CMS Pipelines. It is also relatively straightforward to write PETs for use with XEDIT, using XEDIT subcommands and values returned by the EXTRACT subcommand. These programming tools are generally available with VM/CMS as delivered from IBM and no additional software is required. Documentation on using the basic tools can be found in system help files or in IBM reference manuals. This article will show how these basic tools can be combined to create new PETs.

THE BASIC PROGRAM STRUCTURE

In general, PETs are written with a primary loop. Within the loop, the program displays information in a CMS window and then pauses until the user responds in some fashion. The program can then alter the information displayed on the screen, perform a function, or exit, according to directives specified by the user. Simplistically, the basic steps in these programs are as follows:

- Start the program.
- Define and initialize virtual screens and windows.

- Other initial processing.
- Loop:
  - Display information.
  - Pause and receive keystrokes (or ‘mouse clicks’) from the user.
  - Analyse the keystrokes and cursor position.
  - Perform the requested function (or exit if requested).
  - Update information on the virtual screen.
  - Continue the loop.
- Delete virtual screens and windows (usually).
- Other termination processing.
- End the program.

The approach to programming PETs XEDIT macros varies somewhat from that used to program EXECs. Programming an XEDIT macro might include redefining the ‘meaning’ of the ENTER key, displaying XEDIT reserved lines, and using the EXTRACT subcommand to determine which keystrokes were pressed and the position of the cursor on the screen when the last key was pressed. CMS virtual screens and windows can be used if appropriate.

CMS VIRTUAL SCREENS AND WINDOWS

At the core of all interactive PETs are CMS virtual screens and CMS windows. Virtual screens are writable ‘presentation spaces’ that can contain text intended for display on a 3270 terminal. Conceptually, virtual screens are rectangular spaces which contain lines of text.

Virtual screens *can* be 80 columns wide by 24 lines down, but they need not be; they can be defined with fewer or more than 80 columns and with a variable number of lines – virtual screens with thousands of lines of data are possible.

Each virtual screen is associated with a CMS virtual window. CMS windows are rectangular objects which map the contents of a virtual screen onto a real 3270 display. A window can be equal in size to a real 3270 display, or it may be smaller than a real device.

An open CMS window permits a user to view data on a virtual screen. Figure 1 shows the relationship between a virtual screen containing text and a window which facilitates viewing that text. In some cases, the virtual screen and the window are defined in such a way that:

1 The window shows the entire contents of the virtual screen.

2 The window completely fills a standard 3270 display.

In other cases, the virtual screen is larger than the window (as shown in Figure 1) and the window must be repositioned on the virtual screen in order to view the 'hidden' contents.

Several steps are required to use virtual screens and windows. Each step can be accomplished by issuing one or more CMS command from within a REXX program. The basic steps are as follows:

1 Define the virtual screen size and other attributes.

2 Define the window size and other attributes.

3 Connect the window to the virtual screen.

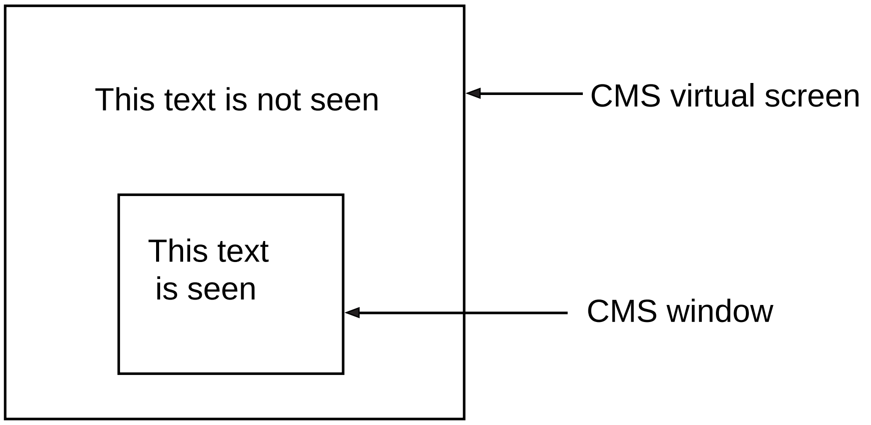
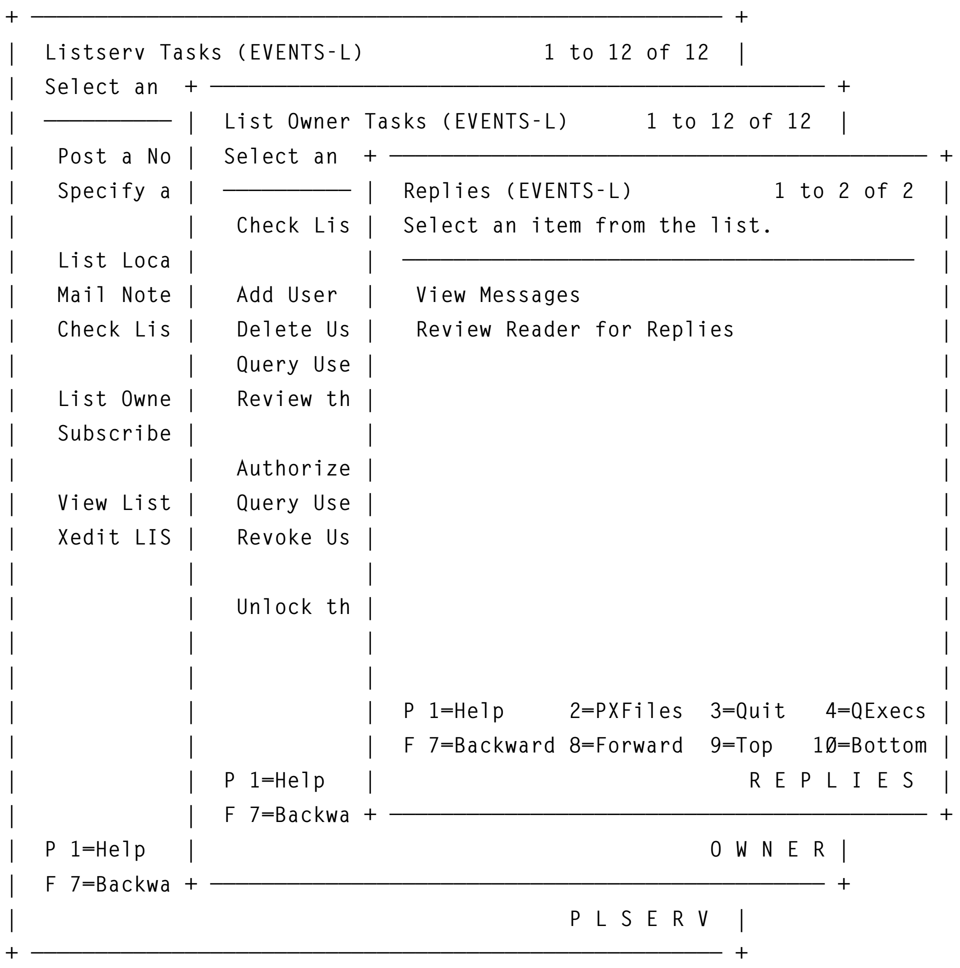


Figure 1: Relationship between a CMS virtual screen and its associated CMS window

4 Write text into the virtual screen.

5 Open the window.

The process becomes a little more complex when more than one virtual screen and more than one window are defined and in use. Windows can be opened or closed, placed in front or behind other windows, etc. Because an application can open several windows simultaneously, some care should be taken to ensure that the result is as usable and user-friendly as possible. Figure 2 shows the results from an application called PLSERV that provides a front-end to Listserv processing (a product of L-Soft International).



```
+ ------------------------------------------------------- +
|  Listserv Tasks (EVENTS-L)              1 to 12 of 12   |
|  Select an  + ------------------------------------------------- +
|  ---------- |  List Owner Tasks (EVENTS-L)      1 to 12 of 12   |
|   Post a No |  Select an  + ------------------------------------------- +
|   Specify a |  ---------- |  Replies (EVENTS-L)           1 to 2 of 2   |
|             |   Check Lis |  Select an item from the list.              |
|   List Loca |             |  ----------------------------------------   |
|   Mail Note |   Add User  |   View Messages                             |
|   Check Lis |   Delete Us |   Review Reader for Replies                 |
|             |   Query Use |                                             |
|   List Owne |   Review th |                                             |
|   Subscribe |             |                                             |
|             |   Authorize |                                             |
|   View List |   Query Use |                                             |
|   Xedit LIS |   Revoke Us |                                             |
|             |             |                                             |
|             |   Unlock th |                                             |
|             |             |                                             |
|             |             |                                             |
|             |             |  P 1=Help     2=PXFiles  3=Quit   4=QExecs  |
|             |             |  F 7=Backward 8=Forward  9=Top   10=Bottom  |
|             |  P 1=Help   |                             R E P L I E S   |
|             |  F 7=Backwa + ------------------------------------------- +
|  P 1=Help   |                                        O W N E R  |
|  F 7=Backwa + ------------------------------------------------- +
|                                           P L S E R V   |
+ ------------------------------------------------------- +
```

Figure 2: Example of simultaneously open CMS windows.

DEFINING THE VIRTUAL SCREEN

A virtual screen is defined by issuing the VSCREEN command. For example:

```
'VSCREEN DEFINE TESTSCRN 20 66 1 2'
```

where:

- ‘DEFINE’ is the VSCREEN command option.
- ‘TESTSCRN’ is the name of the virtual screen.
- ‘20’ is the number of scrollable lines of data in the virtual screen.
- ‘66’ is the number of columns in the virtual screen.
- ‘1’ is the number of ‘reserved lines’ at the top of the virtual screen.
- ‘2’ is the number of ‘reserved lines’ at the bottom of the virtual screen.

Typically, reserved lines are used for non-varying information such as titles or PF key definitions (eg 1=Help). Data lines are intended to be written and possibly rewritten. But there is no strict requirement governing the type of data that can be written to these different areas on a virtual screen. The primary difference seems to be that data lines can be scrolled and reserved lines are fixed in place.

DEFINING A WINDOW

A window is defined by issuing the WINDOW command. For example:

```
'WINDOW DEFINE TESTWIN 2Ø 67 3 7'
```

where:

- ‘DEFINE’ is the WINDOW command option.
- ‘TESTWIN’ is the name of the window.
- ‘20’ is the number of lines in the window.
- ‘67’ is the number of columns in the window.
- ‘3’ specifies that the top row of the window is to be placed on line 3 of a real 3270 display.

- ‘7’ specifies that the leftmost column of the window is to be placed on column 7 of a real 3270 display.

There are, of course, options that can be specified when defining virtual screens and windows. Options alter virtual screen and window attributes such as colour, borders, whether or not data in a window is fixed or scrollable, and so on. For details see the help files ‘HELP VSCREEN DEFINE’ and ‘HELP WINDOW DEFINE’.

CONNECTING A WINDOW TO A VIRTUAL SCREEN

A window is connected to a specific virtual screen with the WINDOW command. For example:

```
'WINDOW SHOW TESTWIN ON TESTSCRN 1 1'
```

where:

- ‘SHOW’ is the WINDOW command option.
- ‘TESTWIN’ is the name of the window.
- ‘TESTSCRN’ is the name of the virtual screen.
- ‘1’ specifies that line 1 of the virtual screen will be shown on the top line of the window.
- ‘1’ specifies that column 1 of the virtual screen will be seen in the leftmost column of the window.

WRITING TEXT TO A VIRTUAL SCREEN

Text is queued up for writing to a virtual screen with the VSCREEN command. For example:

```
'VSCREEN WRITE TESTSCRN 9 1 66 (FIELD Hello, World!'
```

where:

- ‘WRITE’ is the VSCREEN command option.
- ‘TESTSCRN’ is the name of the virtual screen.
- ‘9’ specifies the line in which the text is to be written.

- ‘1’ specifies the column in which the text is to be written.
- ‘66’ specifies the length of the text field to be written.
- ‘FIELD’ is a VSCREEN command option that specifies a field definition.
- ‘Hello, World!’ is the text to be written to the virtual screen.

The length of the data string queued to a virtual screen should not exceed one less than the width of the virtual screen, as specified in the VSCREEN DEFINE command.

Text can be written to a virtual screen with the VSCREEN command (there are other commands as well). For example:

```
'VSCREEN WAITREAD TESTSCRN'
```

where:

- ‘WAITREAD’ is the VSCREEN command option.
- ‘TESTSCRN’ is the name of the virtual screen.

Here, the WAITREAD command option writes any queued data to the virtual screen and then waits for the user to respond. The user can enter some text (if required), but he must press a PF key, a PA key, the CLEAR key, or the ENTER key to terminate the WAITREAD. VSCREEN stores the user-entered text and other information in variables that can then be retrieved by the program.

Virtual screens and windows can be deleted with the appropriate commands. For example:

```
'WINDOW DELETE TESTWIN'
'VSCREEN DELETE TESTSCRN'
```

VSCREEN and WINDOW commands can be included in a REXX program in the usual manner, as follows:

```
/* Example of VSCREEN and WINDOW commands                          */

'VSCREEN DEFINE TESTSCRN 20 66 1 2'     /* Define the virtual screen.*/
'WINDOW DEFINE TESTWIN 20 67 3 7'       /* Define the window.        */

'WINDOW SHOW TESTWIN ON TESTSCRN 1 1'   /* Connect the window to the */
```

```
                                        /* virtual screen.          */

'VSCREEN WRITE TESTSCRN 9 1 66 (FIELD', /* Queue a line of text to   */
    'Hello, World!'                     /* the virtual screen.       */

'VSCREEN WAITREAD TESTSCRN'             /* Update the virtual screen */
                                        /* and await a response.     */

'WINDOW DELETE TESTWIN'                 /* Delete the window.        */
'VSCREEN DELETE TESTSCRN'               /* Delete the virtual screen.*/

Exit
```

WRITING TO AND READING FROM WINDOWS

It may be appropriate to display some information in a window and then close that window without further action. However, many applications lend themselves to repeated interaction with end users.

In such cases, there may be a primary window that displays information and receives text or directives from the end user, and then loops again to refresh the text in the window or to receive additional directives. The sample program that follows employs an appropriate looping structure:

```
/* Looping with a virtual screen                                   */

'VSCREEN DEFINE TESTSCRN 10 36 1 2'
'WINDOW DEFINE TESTWIN 10 37 8 15'
'WINDOW SHOW TESTWIN ON TESTSCRN 1 1'

Do loop = 1 By 1 Until(loop=3)
   Select;
      When loop = 1 Then datastring = 'Hello, World!'
      When loop = 2 Then datastring = 'Second time around.'
      When loop = 3 Then datastring = 'Well, this is it!'
      Otherwise NOP
      End
   'VSCREEN WRITE TESTSCRN 4 1 36 (FIELD' datastring
   'VSCREEN WAITREAD TESTSCRN'
   End loop

'WINDOW DELETE TESTWIN'
'VSCREEN DELETE TESTSCRN'
Exit
```

If a virtual screen is defined with reserved lines at the top and/or bottom, it may be appropriate to add static instructions on those lines as a guide to users. The looping example is extended in the code below to include commands that write a title on the top line of the virtual screen and instructions on the bottom line. Please note that text can be displayed in different colours according to the options specified on the VSCREEN WRITE commands.

```
/* Writing static text on reserved lines                         */

'VSCREEN DEFINE TESTSCRN 10 36 1 2'
'WINDOW DEFINE TESTWIN 10 37 8 15'
'WINDOW SHOW TESTWIN ON TESTSCRN 1 1'

'VSCREEN WRITE TESTSCRN 1 1 36 (RES', /* Queue text to reserved line */
'YELLOW FIELD The World of Windows!'  /* number 1 (the top).         */

'VSCREEN WRITE TESTSCRN -2 1 36 (RES', /* Queue text to the second   */
   'RED FIELD Press ENTER (or click',  /* from the bottom reserved   */
   'your mouse!)'                      /* line (the -2 line).        */

'VSCREEN WRITE TESTSCRN -1 1 36 (RES', /* Queue text to the bottom   */
   'RED FIELD to continue...'          /* reserved line (the -1 line)*/

Do loop = 1 By 1 Until(loop=3)
   Select;
      When loop = 1 Then datastring = 'Hello, World!'
      When loop = 2 Then datastring = 'Second time around.'
      When loop = 3 Then datastring = 'Well, this is it!'
      Otherwise NOP
      End
   'VSCREEN WRITE TESTSCRN 4 1 36 (FIELD' datastring
   'VSCREEN WAITREAD TESTSCRN'
   End loop
'WINDOW DELETE TESTWIN'
'VSCREEN DELETE TESTSCRN'
Exit
```

Reading text from a window requires a user to enter information into an ‘unprotected field’ in a window. Text entered in an unprotected field is stored in a stem variable and can be retrieved by referring to specific elements of that stem variable.

As a convenience, some provision should be made to properly position the cursor so that a user need not spend time fiddling with the arrow or tab keys. As a practical matter, it may be appropriate to alter some of the text on the virtual screen as the process continues.

The EXEC below displays a window, asks the user to enter his name, receives the name, and then redisplays the window with altered text and a new position for the cursor.

```
/* Reading text with a window                                        */

'VSCREEN DEFINE TESTSCRN 10 36 1 2'
'WINDOW DEFINE TESTWIN 10 37 8 15'
'WINDOW SHOW TESTWIN ON TESTSCRN 1 1'

'VSCREEN WRITE TESTSCRN  1 1 36 (RES YELLOW FIELD',
   'Please enter your name.'
'VSCREEN WRITE TESTSCRN -2 1 36 (RES RED FIELD',
   'Press ENTER (or click your mouse!)'
'VSCREEN WRITE TESTSCRN -1 1 36 (RES RED FIELD to continue...'

/* The following lines queue the prompt, queue/define an unprotected  */
/* field to receive the name, set the cursor in the first position of */
/* the unprotected field, refresh the virtual screen and await a      */
/* response from the user.                                            */

'VSCREEN WRITE  TESTSCRN 4  1 11 (PROTECT GREEN FIELD Your name:'
'VSCREEN WRITE  TESTSCRN 4 12 23 (NOPROTECT BLUE FIELD  '
'VSCREEN CURSOR TESTSCRN 4 13 (DATA'
'VSCREEN WAITREAD TESTSCRN'

/* Element WAITREAD.3 contains information about the text which was   */
/* typed into the window, including the line number, column number,   */
/* and specific text.  Parsing out "value" retrieves the user's name. */

Parse Var waitread.3 type ln cn value
name = Strip(value)

/* The following lines queue new text to the virtual screen, place the*/
/* cursor onto a lower reserved line, refresh the screen and await a  */
/* response from the user.                                            */

'VSCREEN WRITE  TESTSCRN 1 1 36 (RES YELLOW FIELD' Left('Thanks!',35)
'VSCREEN WRITE  TESTSCRN 4 1 36 (NOPROTECT FIELD' Left('Hello,'name,35)
'VSCREEN CURSOR TESTSCRN -2 8 (RESERVED'
'VSCREEN WAITREAD TESTSCRN'

'WINDOW DELETE TESTWIN'
'VSCREEN DELETE TESTSCRN'
Exit
```

THE WAITREAD. STEM VARIABLE

The VSCREEN WAITREAD command performs several functions:

1 Virtual screens are refreshed with text previously queued to them.

2 The image displayed on the real 3270 screen is updated.

3 The next interrupt (ENTER, CLEAR, PA or PF key) is awaited.

4 Text entered by the user is retrieved and stored, along with information about which key was pressed and the cursor position, in elements of the WAITREAD. stem variable.

The elements of WAITREAD. contain the following information:

- WAITREAD.0 – the number of elements returned (excluding WAITREAD.0).
- WAITREAD.1 – the specific interrupt key that was pressed.
- WAITREAD.2 – the position of the cursor when the interrupt occurred.
- WAITREAD.3 through to WAITREAD.n – information about fields that were changed; line number, column number, and modified text.

An EXEC can examine the contents of WAITREAD.1 to determine specifically which interrupt key was pressed. For example, if PF Key 3 was pressed, WAITREAD.1 would contain the following string:

```
'PFKEY  3'
```

or if the ENTER key was pressed, WAITREAD.1 would contain the following string:

```
'ENTER'
```

An EXEC can examine the contents of WAITREAD.2 to determine where the cursor was positioned on the virtual screen when the interrupt occurred.

WAITREAD.2 will contain a string similar to this:

```
'CURSOR 3 10 DATA'
```

indicating that the cursor was on line 3, column 10; the virtual screen line was defined as a DATA line rather than a RESERVED line. Or, WAITREAD.2 will contain a string similar to this:

```
'CURSOR 1 40 RESERVED'
```

indicating that the cursor was positioned on reserved line number 1 (the top of the virtual screen) in column 40.

An EXEC can examine the contents of the WAITREAD.3 through WAITREAD.n stem variable elements and retrieve information about virtual screen fields that have been changed. Information about the first changed field (top to bottom, left to right) is stored in WAITREAD.3. If changes were made to a second field on the same virtual screen, then information about the second changed field is stored in WAITREAD.4, and so forth. The value stored in WAITREAD.0 can be examined to determine how many fields were changed (the value in WAITREAD.0 minus 2). WAITREAD.3 and later elements will contain a string similar to this:

```
'DATA 4 10 text which has been entered'
```

indicating that the string ‘text which has been entered’ was found in a changed field, which starts in column 10 on data line 4 of the virtual screen. If the text was changed in an unprotected reserved line, then WAITREAD.3 would contain a string similar to this:

```
'RESERVED 1 3 text which has been entered on a reserved line'
```

The on-line help file can be reviewed for a more detailed description of the WAITREAD. stem variable.

```
HELP VSCREEN WAITREAD
```

By carefully assessing the values returned in the WAITREAD. stem variable elements, the REXX program can determine what text (if any) was entered onto the screen, which interrupt key was pressed, and the position of the cursor when that interrupt key was pressed.

HOW MOUSE CLICKS ARE RECEIVED AND INTERPRETED

From the previous discussions on virtual screens, CMS windows, and the WAITREAD. stem variable, it should be clear that interactive programs can be written that display information in windows and react to user keystrokes. For example, if a user presses PF Key 3, then that fact is passed back to the program through the WAITREAD.1 variable. The program examines the value of WAITREAD.1, finds the string ‘PFKEY 3’, and terminates normally:

```
Do loop = 1 By 1
  .
  .
If Left(waitread.1,8) = 'PFKEY  3' Then Leave loop
  .
  .
End loop
  .
  .
  .
Exit(Ø)
```

Similarly, if a user presses the ENTER key, the value of WAITREAD.1 is updated to contain the string ‘ENTER’. Furthermore, the position of the cursor when the ENTER key is pressed is stored as the value of the WAITREAD.2 variable. By parsing WAITREAD.2, the line and column corresponding to the cursor’s position in the virtual screen can be determined:

```
/* WAITREAD.2 contains a string similar to 'CURSOR 3 1Ø DATA' */
Parse Var WAITREAD.2 . lineno columno area .
```

Therefore, the program can learn the position of the cursor when the ENTER key is pressed, and proceed accordingly.

In many 3270 terminal emulation software packages a mouse action is (or can be) defined to emulate the two actions ‘set cursor’ and ‘press enter’. A single click of the right mouse button, for example, can be configured to emulate setting the 3270 cursor and pressing the ENTER key.

In practice, the PC or workstation pointer is moved with the mouse to some location on the screen, and the right mouse button is clicked. That single click repositions the 3270 cursor in the active virtual screen and sends an interrupt to CMS. CMS passes the information along to the VSCREEN WAITREAD process as previously discussed, and variables WAITREAD.1 and WAITREAD.2 are updated as if the real ENTER key had been pressed. The PETs program logic examines these variables and proceeds according to design.

PETs programs are designed to handle mouse clicks in this manner. At the same time they are designed to respond to standard keystrokes and the normal interrupt keys. By handling both keyboard keystrokes and mouse clicks equally well, PETs programs serve both traditional mainframe users and people who prefer to use a mouse. By exploiting this interesting synergy between the workstation mouse and CMS, PETs can bring a new level of productivity and ease of use to the 3270 world.

A FINAL EXAMPLE

The EXEC presented below, while of limited practical value, combines all the elements discussed in this article: virtual screen, CMS window, infinite loop, WAITREAD processing, analysis of the WAITREAD. stem variable values, functional selection, and screen/window clean-up. In addition, the example shows how error messages might be displayed when appropriate:

```
/* Sample Pointer Enabled Tool - Command Menu                           */

'VSCREEN DEFINE MENUSCRN 8 31 2 2'                  /* define screen */
'WINDOW DEFINE MENUWIN 8 32 8 24'                   /* define window */
'WINDOW SHOW MENUWIN ON MENUSCRN 1 1'               /* connect w->s  */

'VSCREEN WRITE MENUSCRN 1 1  31 (RES PR W FIELD',   /* queue title   */
   Center('Command Menu',29)
'VSCREEN WRITE MENUSCRN 1 1  31 (PR G FIELD Filelist'/* queue line 1  */
'VSCREEN WRITE MENUSCRN 2 1  31 (PR G FIELD Help'   /* queue line 2  */
'VSCREEN WRITE MENUSCRN 3 1  31 (PR G FIELD RdrList' /* queue line 3  */
'VSCREEN WRITE MENUSCRN 4 1  31 (PR G FIELD SendFile'/* queue line 4  */
'VSCREEN WRITE MENUSCRN -1 1 31 (RES PR R FIELD',   /* queue help    */
   'Click on a command.  PF3=Quit'

message = ''                                        /* init error msg*/
Do loop = 1 By 1                                    /* loop forever  */
   'VSCREEN WRITE MENUSCRN 2 1 31 (RES PR Y FIELD', /* queue err msg */
      Left(message,29)
   'VSCREEN CURSOR MENUSCRN 1 1 (DATA'              /* set cursor    */
   'VSCREEN WAITREAD MENUSCRN'                      /* refresh screen*/
   message = ''                                     /* clear err msg */
   keystroke = Left(waitread.1,8)                   /* get keystroke */
   Parse Var waitread.2 . ln cn area .              /* get line numb */
   Select;
      When keystroke = 'PFKEY  3' Then Leave loop   /* pf3 pressed?  */
      When ln = -1 & cn = -1 Then Leave loop        /* outside win?  */
      When area ¬= 'DATA'                            /* data area?    */
         Then message = 'Incorrect selection'
      When ln = 1 Then 'EXEC FILELIST'               /* line 1?       */
      When ln = 2 Then 'HELP'                        /* line 2?       */
      When ln = 3 Then 'EXEC RDRLIST'                /* line 3?       */
      When ln = 4 Then 'EXEC SENDFILE'               /* line 4?       */
      Otherwise message = 'Unknown option'           /* set err msg   */
      End
   End                                               /* continue loop */

'WINDOW DELETE MENUWIN'                              /* delete window */
'VSCREEN DELETE MENUSCRN'                            /* delete screen */
Exit(0)                                              /* end EXEC      */
```

FURTHER INFORMATION

Further information about the PETs project can be found at the following Web location: http://vm.uconn.edu/~pets/.

Editor's note: in a future article, the author will discuss mouse-clickable enhancements to XEDIT.

Richard G Ellis
Director, Computing and Information Systems
University of Connecticut (USA)

The DIRMAINT Synchronous Application Interface

Until recently, the Directory Maintenance Program Product (DIRMAINT) has had an unsatisfactory programming interface. Programs interacting with DIRMAINT have had to wait for messages from the server, and then analyse text that contained a random mixture of constant and variable data.

With Release 1.5, a new Synchronous Application Interface (SAPI) has been introduced. This is briefly introduced as a GUPI in a 3-page Appendix C to the *Command Reference* manual, but there is very little explanation of how the interface should be used.

What has been provided in the SAPI interface is support for a new ‘language’ for DIRMAINT messages. Alongside the default AMENG, UCENG, and KANJI, there is now 1SAPI. If a DIRMAINT command is issued with 1SAPI as the active language, communication between the caller and the server uses SMSG and IUCV, and all responses will be returned to the caller in one of two fixed formats. The standard format is:

```
DVHrtnnnnnI REQUEST=number RTN=DVHrtn MSG=nnnn FMT=nn
SUBS= any number of tokens to be substituted in the message skeleton
```

If the last character of the ‘SUBS=’ string is a comma (which may appear in the middle of a word), the message is followed by the second format:

```
     DVHmodnnnnI CONT=the rest of the string
```

Note that there is a single blank after ‘SUBS=’, but not after any of the other ‘=’ keywords. With this fixed-format message pattern, it is much easier to find the keywords required by the calling program, and decide what action is needed next.

Appendix C describes ‘two sample programs’, DIRMSAPI and DVHSAPI. In fact, only DIRMSAPI is a sample. If it is renamed to filetype EXEC, it can be used from the console with exactly the same syntax as the standard DIRMAINT EXEC, but its main use is to demonstrate the interface with the DVHSAPI EXEC. This one is not a sample, but a supported part of the DIRMAINT product – there have even been APARs taken against it and fixed. It is designed to be used only as a subroutine – that is why the DIRMSAPI sample is provided. The user interface disk has two versions of the DVHSAPI EXEC, in source and compiled form. The compiled version is obviously preferable for production use, since it performs better, but the source version has been put through the EXECUPDT process before release, and all comments and indentation stripped out. So it is very difficult to use as a tool to understand the interface. For that, you need to go to the version on the maintenance disk. This is well commented, and gives you a good idea of the broad pattern of the process.

It has to be said that this process is very complex, since a lot of DIRMAINT’s server processing is asynchronous, and therefore unsuited to a synchronous interface. The basic flow of DVHSAPI is as follows:

1 Set up the Globalv values needed for the SAPI interface.

2 Call the DIRMAINT EXEC with the command string.

3 Call WAKEUP (distributed with the DIRMAINT product as DVHWAKE) to wait for incoming responses arriving in SMSGs.

4 Store them in a stem variable (stem DVHSAPI.).

5 When the final message has been received, or when WAKEUP times out, it stores the DVHSAPI. stem variables in the calling EXEC, and resets the original environment.

DVHSAPI is governed by state codes, which change as the forecast messages are received. Only when the final message arrives will the EXEC return to the caller with the appropriate return code. Within the flow, there are a lot of complications, particularly those caused by deleting mini-disks. Whereas most successful transactions end with message DVHREQ2289I, when a disk is deleted, the DIRMAINT server only does the preliminary work. The disk is then transferred to a DATAMOVE machine (with an internal TMDISK command), and a later series of messages reports the progress of the DLINK and ZAPMDISK phases. (DLINK deletes Link records to the disk from other user-ids, and ZAPMDISK finally deletes the disks and returns the extent to the free pool.)

Since it is impossible to run a ZAPMDISK while any user has a link to the disk, this means that the final messages for some DMDISK commands can arrive long after the rest of the transaction has completed. (The longest delay I have seen so far is six weeks, when somebody tried to delete some SQL database disks without stopping the server.)

It is because of this potential delay that WAKEUP is programmed to time out. Control is returned to the calling EXEC within a reasonable time, and it can process all the messages that have arrived so far. However, any messages arriving later will be stored in the IUCV buffer, and appear at the top of the messages from the next transaction. You need to bear this in mind when designing the calling EXEC.

Appendix C is mostly made up of a section entitled *Applied SAPI Coding Rules*. The first and third bullet points have been overtaken by later PTFs, so they need major modification.

The first bullet point discusses the need to issue:

```
     EXEC DIRMAINT EXECLOAD
```

before issuing multiple DIRMAINT commands. If your caller is a long-running application that is likely to continue across a restart of the DIRMAINT server machine, it is essential that you have applied the latest service. There can be an I/O error reading the WHERETO DATADVH file on the user interface disk after a server restart. This disk is permanently accessed by the EXECLOAD command, whereas it is accessed and released for each individual command if the EXECLOAD command has not been issued. A new PTF exploits update-in-place for the file, which is rewritten as part of server initialization, so that a user already accessing the disk will see the changes to the file without reaccessing it.

It also suggests issuing an:

```
     EXEC DIRMAINT EXECDROP
```

before the EXECLOAD. This is no longer necessary, since EXECLOAD does it every time.

The third bullet point discusses return codes. There have been major revisions to several of these, as well as some new messages for severe errors, as a result of a new PTF for APAR VM61741. This was not included in RSU9801, the latest at the time of writing. However, that RSU also includes a lot of changes that enhance performance, and I recommend installing it, or any later level.

The 1SAPI messages, although ideal for applications, are very difficult for an ordinary reader to interpret. If you need to display or print the messages, they can be translated to the normal format by calling DVHMSG with your default language instead of 1SAPI. Specimen code to do this is in the DIRMSAPI sample.

To sum up: the new SAPI interface makes it much easier to write programs that depend on DIRMAINT messages, but you need to analyse the DIRMSAPI sample carefully to see how to handle the new message format. You should also be aware that delayed responses may come into IUCV long after the main transaction has completed.

Alan Hakim
Hikmet (VM) Ltd (UK)

VM news

IBM has announced Automated Unix System Option for VM, VSE, and OS/390. Providing an automated Unix application platform, it is designed not to require OS/390 (MVS) skills for management or maintenance. Typical OS/390 Unix-based applications and enablers include Web content hosting; e-business applications; Lotus Domino; and Java applications and applets.

For further information contact your local IBM representative.

* * *

VM users can benefit from BOS-complement from Open Software Technologies, a context sensitive on-line help and application documentation system for VM, MVS, and VSE.

Users can interactively create pop-up help windows and on-line application documentation for mainframe applications running under VTAM. BOS-complement windows are integrated into applications and immediately accessible by a PA or PF hotkey without programming changes or compiles. There is an import function for text documents. System tables, VSAM files, and all types of database can be directly accessed for on-line help display.

For further information contact:
Open Software Technologies, 1230 Douglas Avenue, 300 Longwood, FL 32779, USA.
Tel: (407) 788 7173.
URL: http://www.open-softech.com.

VM users can benefit from the Workstation Group's netCONVERT, a cross-platform data conversion utility designed to convert data between IBM mainframe and Unix formats, as well as to support cross-platform migration projects.

netCONVERT can run on VM and MVS in addition to the major flavours of Unix. Features in Version 2.10 include support for mainframe F, FB, V, VB, and VBS formats; direct read and write support for VSAM files; support for ANSI fixed, variable, and segmented record types, FORTRAN, MicroFocus COBOL, text, and CSV; tape input and output in IBM and ANSI label formats; and a test data generator.

For further information contact:
The Workstation Group, 1900 North Roselle Road, Suite 408, Schaumburg, IL 60195, USA.
Tel: (847) 781 6940.
URL: http://www.wrkgrp.com.

* * *

IBM has announced Version 2 of its COBOL and CICS Command Level Conversion Aid (CCCA) for VM. Now a program product, CCCA for VM Version 2 is designed to help convert old COBOL source code to new versions of COBOL. Also new in Version 2 is the capability to convert COBOL applications to use the new IBM Millennium Language Extensions.

For further information contact your local IBM representative.