216

MVS update

September 2004

In this issue

Published by

Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

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

Editor

Trevor Eddolls
E-mail: trevore@xephon.com

Publisher

Nicole Thomas
E-mail: nicole@xephon.com

Subscriptions and back-issues

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

MVS Update on-line

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

Contributions

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

Printed in England.

Restart JCL

Job Control Language (JCL) is the language that controls OS/390 and z/OS. If you're working as a programmer or operator on a mainframe that runs one of these operating systems, you need to know JCL. This article describes only RESTART JCL. When a program abends or the system fails, you may need to restart a job from the point where the failure occurred. It's possible to restart a job at a point other than the beginning – you can restart from a designated step. This is a step restart using the RESTART parameter of the JOB statement. I have created an edit macro, SUBR, which supports the RESTART mechanism. The SUBR command is used to find EXEC statements in a JCL stream. The EXEC statements are found and presented on a selection list, where you may select the point you want to RESTART from. The JCL job is then modified so that if you SUBMIT it, it will RESTART from the selected point.

I have also created an edit macro, SCAN (like the FIND edit macro), that finds all search words (items) in an open ISPF member. The search words are found and presented on a selection list, where you may select the point (row number). The SCAN macro will position the cursor on the selected row. The SCAN macro requires an input parameter (the search word).

SUBR (AN EDIT MACRO)

```
/* REXX */
/* Step 1                                                  */
/* Edit macro subr restarts batch job control.             */
/* If you don't know the sequence number, you can enter    */
/* just the SUBR command and the macro will display a selection  */
/* list. Select the appropriate sequence number from       */
/* the list. The RESTART sequence number (step id) will    */
/* be added to the RESTART parameter in the batch JCL.     */
/* Step 2                                                  */
/* Enter SUBMIT in the command area and job will resume    */
/* execution at the desired point.                         */
/* trace r */
isredit macro
```

```
Call Job_Check
/* Find all EXEC items in JCL                              */
address ispexec
'isredit seek " EXEC " nx all 1 72'
/* Save count for EXEC                                     */
address ispexec
'isredit (seekc) = seek_counts'
address ispexec 'tbcreate subrt names(rrow, step, desc)'
Call Init
do i=1 to seekc
   /* Find next EXEC item                                  */
   address ispexec
   'isredit find 1 72 next "EXEC"'
   /* Set values for row and col                           */
   address ispexec
   'isredit (row,col) = cursor'
   rrow = row + Ø
   rrow = right(rrow,5)
   /* Set line value                                       */
   address ispexec
   'isredit (line) = line' row
   step = substr(strip(word(line,1)),3)
   desc = strip(substr(line,col))
   address ispexec 'tbadd subrt'
end
Title='Restart Utility'
address ispexec 'addpop row(4) column(17)'
address ispexec 'tbtop subrt'
address ispexec 'tbdispl subrt panel(subrp)'
address ispexec 'tbend subrt'
address ispexec 'rempop all'
Call Check_Restart
if rt=Ø then do
   if cmd='s' | cmd='S' then do
      /* Insert RESTART row with selected stepid          */
      address ispexec
      "isredit line_after 1 = '//            RESTART="step",'"
   end
   else do
      /* Insert RESTART=(*) row                           */
      step='*'
      address ispexec
      "isredit line_after 1 = '//            RESTART=*,'"
   end
   address ispexec
   zedsmsg = 'RESTART '||step
   zedlmsg = 'JCL RESTART '||step||' shown'
   "setmsg msg(isrzØØ1)"
end
address ispexec
```

```
'isredit cursor = 1 1'
address ispexec
'isredit reset'
exit Ø
Job_Check:
  /* Find number of Jobs in batch stream                    */
  address ispexec
  'isredit seek " JOB " nx all 1 72'
  /* Save count for JOB                                     */
  address ispexec
  'isredit (seekj) = seek_counts'
  seekj = seekj*1
  if seekj > 1 then do
     address ispexec
     zedsmsg = 'Warning: '||seekj||' Jobs'
     zedlmsg = 'The Batch stream has '||seekj||' Jobs'
     "setmsg msg(isrzØØ1)"
     Exit
  end
Return Ø
Init:
  /* Set all steps RESTART=*                                */
  rrow = right(Ø,5)
  step = '(*)'
  desc = All steps
  address ispexec 'tbadd subrt'
Return Ø
Check_Restart:
  rt = Ø
  /* Find all RESTART items in JCL                          */
  address ispexec
  'isredit seek "RESTART" nx all 1 72'
  /* Save count for RESTART                                 */
  address ispexec
  'isredit (seekr) = seek_counts'
  seekr = seekr*1
  if seekr > 1 then do
     address ispexec
     zedsmsg = 'RESTART error'
     zedlmsg = 'RESTART item occurs '||seekr||' times'
     "setmsg msg(isrzØØ1)"
     rt=8
  end
  if seekr = Ø then rt=Ø
  if seekr = 1 then do
     /* Change old RESTART parameter with new               */
     /* Find RESTART item                                   */
     address ispexec
     'isredit find 1 72 next "RESTART"'
     /* Set values for row and col                          */
```

```
      address ispexec
      'isredit (row,col) = cursor'
      col=col+8
      /* Set line value                                        */
      address ispexec
      'isredit (line) = line' row
      /* Find old RESTART parameter                            */
      osid = substr(line,col)
      osid = translate(osid,' ',',')
      osid = word(osid,1)
      nstep=step
      if osid='*' then do
         osid='RESTART=*'
         nstep='RESTART='||step
      end
      if osid='(*)' then do
         osid='RESTART=(*)'
         nstep='RESTART=('||step||')'
      end
      /* Change old RESTART parameter with new                 */
      isredit change 5 72 osid nstep all
      address ispexec
      zedsmsg = 'RESTART '||step
      zedlmsg = 'JCL RESTART '||step||' shown'
      "setmsg msg(isrz001)"
      rt=4
  end
Return rt
```

SCAN (AN EDIT MACRO)

```
/* REXX */
/* Edit macro scan finds all text strings in a member    */
/* and displays a selection list.                        */
/* Select the appropriate sequence number from the list. */
/* The scan macro will position the cursor to the        */
/* selected row.                                         */
/* trace r */
'isredit macro (item)'
parse var item
/* Check input item - search word                        */
if item='' then do
   address ispexec
   zedsmsg = 'Search word'
   zedlmsg = 'Search word is missing'
   "setmsg msg(isrz001)"
   exit
end
/* Find all items                                        */
```

```
address ispexec
'isredit seek "'item'" nx all'
/* Save count for search word - item                        */
address ispexec
'isredit (seekc) = seek_counts'
address ispexec 'tbcreate scant names(rrow, desc)'
do i=1 to seekc
   /* Set values for row and col                            */
   address ispexec
   'isredit (row,col) = cursor'
   rrow = row + 0
   rrow = right(rrow,5)
   /* Set line value                                        */
   address ispexec
   'isredit (line) = line' row
   desc = strip(substr(line,col))
   address ispexec 'tbadd scant'
   /* Find next item                                        */
   address ispexec
   'isredit find next "'item'"'
end
/* Search word not found                                    */
if seekc=0 then do
   address ispexec
   zedsmsg = item||' not found'
   zedlmsg = item||' not found'
   "setmsg msg(isrz001)"
end
/* Search word found                                        */
else do
   Title='Scan Utility'
   address ispexec 'addpop row(4) column(17)'
   address ispexec 'tbtop scant'
   address ispexec 'tbdispl scant panel(scanp)'
   if cmd='s' | cmd='S'
   then row = rrow
   else row = 1
   address ispexec 'tbend scant'
   address ispexec 'rempop all'
   if rt=0 then do
      address ispexec
      zedsmsg = 'Find '||item
      zedlmsg = 'Find '||item
      "setmsg msg(isrz001)"
   end
   address ispexec
   'isredit cursor = 'row' 0'
   address ispexec
   'isredit reset'
```

```
end
exit Ø
```

SUBRP (A PANEL)

```
)Attr Default(%+_)
   ! type(text)   intens(high) caps(on ) color(yellow)
   $ type(output) intens(high) caps(off) color(yellow)
   ? type(text)   intens(high) caps(on ) color(green) hilite(reverse)
   # type(text)   intens(high) caps(off) hilite(reverse)
   ] type( input) intens(high) caps(on ) just(left ) pad('_')
   ^ type(output) intens(low ) caps(off) just(asis ) color(green)
)Body Window(57,13) Expand (//)
%-/-/- ? Selection Result +%-/-/-
%Command ===>_zcmd                  / /%Scroll ===>_amt +
+-----------------------------------------------------
+cmd:!S+           $title                             +
+-----------------------------------------------------
#cmd#Line #Step    #Description                       +
)Model
 ]z+^z    ^z       ^z                                 +
)Init
  .ZVARS = '(cmd rrow step desc)'
  &amt = PAGE
  &cmd = ''
)Reinit
)Proc
)End
```

SCANP (A PANEL)

```
)Attr Default(%+_)
   ! type(text)   intens(high) caps(on ) color(yellow)
   $ type(output) intens(high) caps(off) color(yellow)
   ? type(text)   intens(high) caps(on ) color(green) hilite(reverse)
   # type(text)   intens(high) caps(off) hilite(reverse)
   ] type( input) intens(high) caps(on ) just(left ) pad('_')
   ^ type(output) intens(low ) caps(off) just(asis ) color(green)
)Body Window(57,13) Expand (//)
%-/-/- ? Selection Result +%-/-/-
%Command ===>_zcmd                  / /%Scroll ===>_amt +
+-----------------------------------------------------
+cmd:!S+           $title                             +
+-----------------------------------------------------
#cmd#Line #Description                                +
)Model
 ]z+^z    ^z                                         +
)Init
```

```
  .ZVARS = '(cmd rrow desc)'
  &amt = PAGE
  &cmd = ''
)Reinit
)Proc
)End
```

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

Retrieve SMS information and DASD usage statistics using a REXX tool

The purpose of the tool DSINFO (DataSet INFOrmation) is to generate a report giving SMS (Storage Management Subsystem) information about a set of datasets. This tool also gives information about the total DASD (Direct Access Storage Device) space used by a user(s)/set of datasets.

This information is required for analysis and design of procedures/ processes with the objective of performance tuning. A few reasons for using this tool are listed below:

1 Classification of datasets based on their storage unit (DASD/ TAPE).

2 Capacity planning for the processes (individual files/total DASD space used).

3 List all datasets catalogued by a user under his user-id along with SMS information.

4 Details like storage unit, space allocated, dataset organization, record length, and record format of given datasets could be obtained in one shot for miscellaneous datasets.

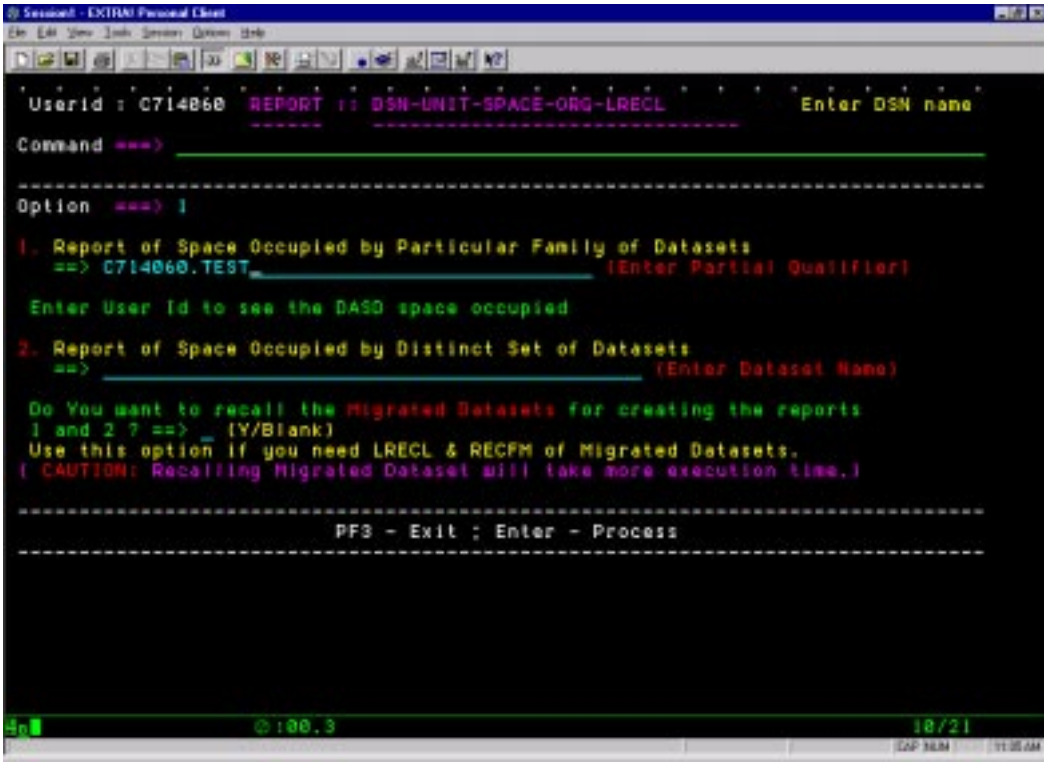


Figure 1: Selecting datasets starting with a particular prefix

TOOL DETAILS

The report could be generated using either of two options.

The first option is to get a report for all the datasets starting with a particular prefix. For example:

```
C714060.TEST
```

This will create a report for all the datasets starting with C714060.TEST.*. See Figure 1.

The second option is to get a report for all the miscellaneous datasets given by a user. The user needs to create a master dataset before executing the tool. This master dataset must contain the names of all the miscellaneous datasets. This

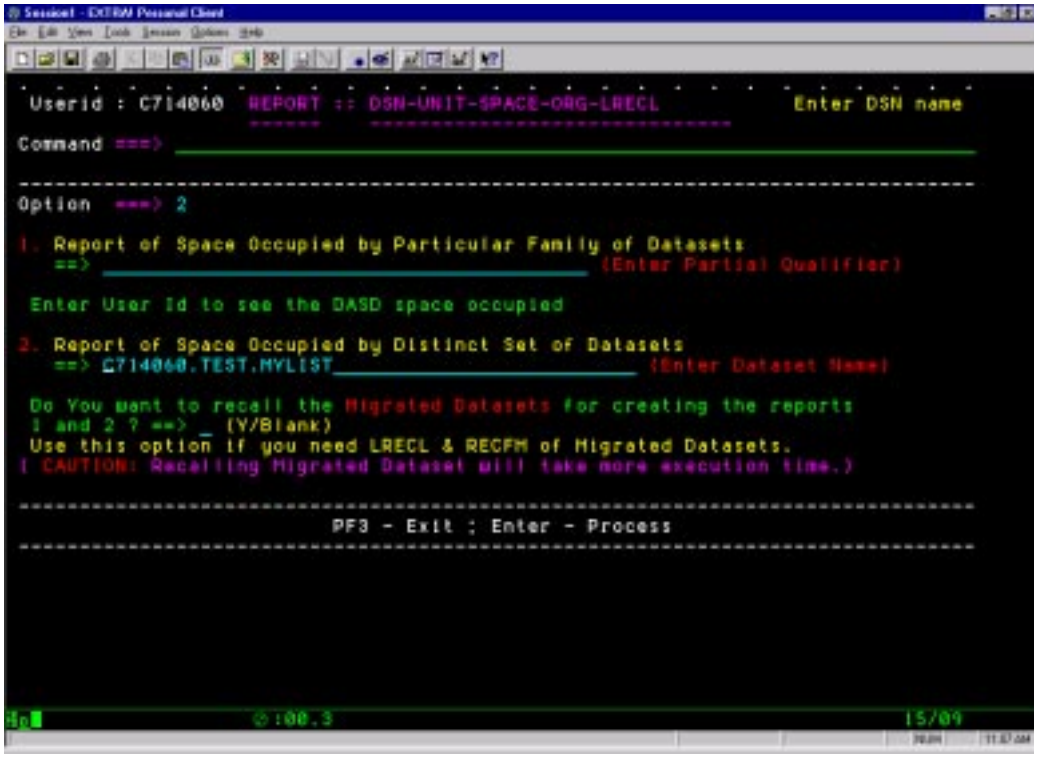


Figure 2: Miscellaneous dataset list selection

dataset can have any name. This master dataset will be used as input to the DSINFO tool. For example, sequential file C714060.TEST.MYLIST will contain a list of datasets for which SMS information is required. See Figure 2.

The tool will automatically create the report dataset USERID.TEST.REPORT.DSSIZE, where USERID is the user's mainframe user-id. If the report dataset is already catalogued, the tool will overwrite the contents of the report. (To save the reports generated with different input, copy the dataset USERID.TEST.REPORT.DSSIZE into another dataset.)

The tool provides an option of creating a report with or without calling migrated datasets. The default option is blank (ie no recall of migrated datasets). If you specify 'Y', all the migrated datasets will be recalled and it may take more time to create the report.

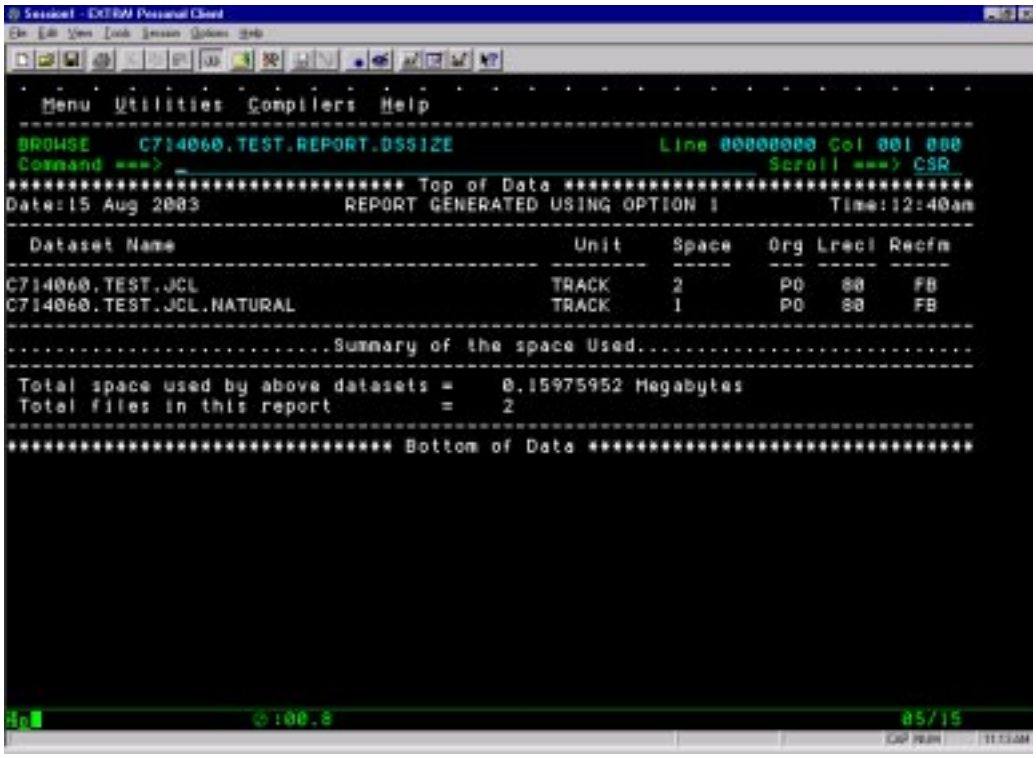


Figure 3: A typical report

Even if the dataset is migrated, its SMS information is obtained, though only partially. Details like dataset storage unit, space, and organization are reported. The information that is not listed is record length and record format. So if record format and length are not important for a particular analysis, having the recall option blank would be a wise decision. This way, the tool would be used optimally.

If the tool is not able to decipher the complete information about the dataset SMS information, it may throw a few messages on the screen. There is no problem with such messages. The tool will continue working fine.

A typical report is shown in Figure 3.

HOW TO USE THE TOOL

Upload the REXX tool DSINFO as a member of a CLIST library or EXEC library and concatenate it with SYSPROC or SYSEXEC respectively. This depends on your mainframe default installations and may vary from one mainframe to another.

Upload panel REPORT1 as a member in USERID.TEST.ISPPLIB. If this library is not available then create this library with a record format of FB, record length of 80 bytes, and dataset organization of PDS. The USERID is the user's mainframe user-id.

Then go to the ISPF command shell (to use TSO commands). Just type DSINFO and the panel REPORT1 will pop up. Enter your options and press *Enter* to create the report.

Limitations:

1 For incorrect datasets, it will give proper error messages and at times it may throw the user out of the panel. Please correct the input and reuse the tool.

2 For datasets residing on tape, the tool gives partial SMS information. However, it mentions the storage unit as TAPE.

3 For VSAM datasets, the base along with data and index files will be shown in the report. The complete SMS information about VSAM datasets will be indicated against the VSAM base. Information about the data and index part will remain blanks/nulls. Note that, in such cases, there is no loss of SMS information.

DSINFO

```
/******************************** REXX ********************************/
/*** Purpose: Generate a report giving details of storage unit,    ***/
/***        : space allocated, dataset organization, record length,***/
/***        : and record format for given datasets.                ***/
/*** Input  : Option 1: Get the report for all the datasets starting**/
/***        : with a particular prefix                             ***/
/***        : Option 2: Get the report for all the datasets required**/
/***        : by user. User needs to create a dataset before        **/
/***        : executing the tool. This dataset must contain the    ***/
```

```
/***        : names of specific DSN.                                    ***/
/*** Output : The report with the above stated details.                 ***/
/*Execution : Run the macro in TSO by entering macro name.             ***/
/***                                                                    ***/
/*** Author : Yash (Jun 21, 2003) - Longest Day of the Year            ***/
/************************************************************************/
/***                  Modification Log                                  ***/
/***-----------------------------------------------------------------***/
/************************************************************************/
 panelpds = "'"||userid()||".TEST.ISPPLIB"||"'"
"Ispexec libdef ispplib dataset id ("panelpds")"
Do forever
   "Ispexec display PANEL(report1) CURSOR(y)"
   IF rc >  4  then do
      exit
   end
   "ispexec vget (y op1 op2 v keypress) profile"
   IF keypress = "" then do
     upper op1 op2 v keypress
     X = MSG('OFF')
 /*   trace ?i */
     validation_done = TRUE
     If validation_done = TRUE then do
       If (y ¬= 1) & ( y ¬= 2) then do
          zedsmsg = "Option Incorrect"
          zedlmsg = "Option must be either 1/ 2"
          "ispexec setmsg msg(isrz001)"
          validation_done = FALSE
       END
     END
     If validation_done = TRUE then do
       If (v ¬= 'Y') & (v ¬= "") then do
           zedsmsg = "Recall Option Incorrect"
           zedlmsg = "Recall Option = Y or Blank"
           "ispexec setmsg msg(isrz001)"
           validation_done = FALSE
       END
     END
     If validation_done = TRUE then do
         IF (y = 1) & (op1 = ' ') then do
               zedsmsg = "Enter DSN name"
               zedlmsg = "Enter the DSN for Option 1"
               "ispexec setmsg msg(isrz001)"
               validation_done = FALSE
          END
     END
     If validation_done = TRUE then do
         IF (y = 2) & (op2 = ' ') then do
               zedsmsg = "Enter DSN name"
               zedlmsg = "Enter the DSN for Option 2"
```

```
               "ispexec setmsg msg(isrz001)"
               validation_done = FALSE
            END
     END
     If validation_done = TRUE then do
         IF y = 1 then indsn = op1
         If y = 2 then indsn = op2
 parse var indsn v1 '.' v2 '.' v3 '.' v4 '.' v5 '.' v6 '.' v7 '.' v8
      If (length(v1) > 8) | (length(v2) > 8) | (length(v3) > 8) |,
         (length(v4) > 8) | (length(v5) > 8) | (length(v6) > 8) |,
         (length(v7) > 8) | (length(v8) > 8) | (length(v9) > 8)
         then  validation_done = FALSE
       If validation_done = FALSE then do
         zedsmsg = "Incorrect DSN"
         zedlmsg = "DSN qualifier has length greater than 8 bytes"
         "ispexec setmsg msg(isrz001)"
       END
     END
     If validation_done = TRUE then do
       Select
         When y = 1 then do
                         call Generate_report_1
                         call Write_the_report
                         end
         When y = 2 then do
                         call Generate_report_2
                         call Write_the_report
                         end
         Otherwise nop
       End /* select */
     END /* validation_done = TRUE */
   End  /* keypress = "" */
End /* Do forever */
Exit 0
Generate_report_1:
/*-----------------------------------------------------------------*/
/* Delete & Create the file temporary dataset storage file        */
/*-----------------------------------------------------------------*/
gatdsn = "'"||userid()||".test.gather.dsn"||"'"
"DELETE "|| gatdsn
IF SYSDSN(gatdsn) <> 'OK' THEN DO
   "ALLOCATE DA("gatdsn") NEW SPACE(30,20) TRACK LRECL(80)
    FILE(file1) RECFM(F,B) BLKSIZE(27920) UNIT(sysda)"
END
/*-----------------------------------------------------------------*/
/* Gather the full dataset name that is catalogued in the system*/
/*-----------------------------------------------------------------*/
"ISPEXEC LMDINIT LISTID(ID1) LEVEL("indsn")"
"ISPEXEC LMDLIST LISTID("ID1") DATASET(DSVAR)"
COUNT = 0
```

```
DO WHILE RC = Ø
  COUNT = COUNT + 1
  record.COUNT = DSVAR
  "ISPEXEC LMDLIST LISTID("ID1") DATASET(DSVAR)"
END
If COUNT = Ø then do
  zedsmsg = "Incorrect Partial DSN"
  zedlmsg = "Partial Qualifier does not match any dataset"
  "ispexec setmsg msg(isrzØØ1)"
  return
END
/*-------------------------------------------------------------*/
/* Write all the dataset names in the temporary sequential file */
/*-------------------------------------------------------------*/
"ALLOC FI(file1) DS ("gatdsn")"
"EXECIO * DISKW file1 (STEM record. FINIS"
"FREE FILE(file1)"
/*-------------------------------------------------------------*/
/*   process the datasets stored in a sequential file          */
/*-------------------------------------------------------------*/
call Main_processing
return
Generate_report_2:
/*-------------------------------------------------------------*/
/*   process the datasets supplied by user                     */
/*-------------------------------------------------------------*/
gatdsn = "'"||indsn||"'"
IF SYSDSN(gatdsn) ¬= 'OK' THEN DO
   zedsmsg = "Invalid DSN"
   zedlmsg = "Enter the correct Dataset Name"
   "ispexec setmsg msg(isrzØØ1)"
   exit Ø
End
call Main_processing
return
Main_processing:
/*-------------------------------------------------------------*/
/*   Main logic of the tool                                    */
/*-------------------------------------------------------------*/
repdate = "Date:"|| DATE()
reptime = "Time:"|| TIME('C')
Reptitle1 = 'REPORT GENERATED USING OPTION '|| y
queue left(repdate,2Ø) center(Reptitle1,46) left(reptime,12)
queue left('',8Ø,'-' )
line1A  = "  Dataset Name                                "
line2A  = "   Unit  " " Space  " "Org" "Lrecl" "Recfm"
line1B  = "-------------------------------------------"
line2B  = " --------" " ---- " "--" "----" "----"
line3   = line1A || line2A
line4   = line1B || line2B
```

```
queue line3
queue line4
spaceinBYTES = Ø
"ALLOC FI(file2) DS("gatdsn")"
"EXECIO * DISKR file2 (STEM IN. FINIS"
DO I=1 TO IN.Ø
  PARSE VAR IN.I dsn1 " " JUNK
  file = "'"|| dsn1 || "'"
   If v = 'Y' then
     x = listdsi(file recall)
   else
     x = listdsi(file norecall)
   Select
     When sysreason = Ø then nop /* sysreason Ø ==> DASD  */
     When sysreason = 1 then do  /* sysreason 1 ==> Invalid DSN */
       sysunits   = 'INVALID '
       sysalloc   = Ø
       sysblksize = Ø
       sysblkstrk = Ø
       systrkscyl = Ø
       sysdsorg   = '    '
       syslrecl   = '     '
       sysrecfm   = '      '
     End
     When sysreason = 5 then do  /* sysreason 5 ==> Dataset not */
       sysunits   = 'UNCATLOG'    /* catalouged                  */
       sysalloc   = Ø
       sysblksize = Ø
       sysblkstrk = Ø
       systrkscyl = Ø
       sysdsorg   = '    '
       syslrecl   = '     '
       sysrecfm   = '      '
     End
     When sysreason = 8 then do  /* sysreason 8 ==> Tape       */
       sysunits   = 'TAPE    '
       sysalloc   = Ø
       sysblksize = Ø
       sysblkstrk = Ø
       systrkscyl = Ø
       sysdsorg   = '    '
       syslrecl   = '     '
       sysrecfm   = '      '
     End
     When (sysreason = 9) | (sysreason = 12) then do
       /* sysreason 9  ==>  Migrated  */
       /* sysreason 12 ==>  VSAM  */
       tempfile = "'"||userid()||".temp.hilst.dsn"||"'"
       /* HLIST command automatically allocates tempfile */
       "DELETE "|| tempfile
```

```
    "HLIST dsname("||"'"dsn1||"'"||") ODS("tempfile")"
    "ALLOC FI(file3) DS("tempfile")"
    "EXECIO * DISKR file3 (STEM ab. FINIS"
    "FREE FILE(file3)"
     migtrk = substr(ab.6,75,6) /* track info is at 75th position */
     migorg = substr(ab.6,96,2) /* Org info is at 96th position */
     migtrk = strip(migtrk,'L','Ø')
     migorg = strip(migorg)
     If migtrk = ' ' then do
        migtrk   = Ø
        migtrk   = Ø
        sysalloc = Ø
     End
     sysunits   = 'TRACK'
     syslrecl   = '     '
     sysrecfm   = '      '
     If (sysalloc = '????') | (sysalloc = ' ') then do
        sysalloc   = Ø
        sysblksize = Ø
        sysblkstrk = Ø
        systrkscyl = Ø
        sysdsorg   = '    '
     end
     else do
       sysalloc   = migtrk * 55840   /* 1 trk = 55840 bytes */
       sysdsorg   = migorg
       sysblksize = 1
       sysblkstrk = 1
     End
   End
   When sysreason = 25 then do /* sysreason 25 ==> Unknown storage */
     sysunits = 'Unknown '
     sysalloc   = Ø
     sysblksize = Ø
     sysblkstrk = Ø
     systrkscyl = Ø
     sysdsorg   = '    '
     syslrecl   = '     '
     sysrecfm   = '      '
   End
   Otherwise nop
 END /*select*/
     queue left(dsn1,44) left(sysunits,9) left(sysalloc,8),
           left(sysdsorg,4) left(syslrecl,5) left(sysrecfm,6)
     If sysunits = 'BLOCK' then
       spaceinBYTES = spaceinBYTES + sysalloc * sysblksize
     If sysunits = 'TRACK' then do
       bytesPerTrack = sysblksize * sysblkstrk
       spaceinBYTES = spaceinBYTES + sysalloc * bytesPerTrack
     End
```

```
       If sysunits = 'CYLINDER' then do
         bytesPerCylinder = sysblksize * sysblkstrk * systrkscyl
         spaceinBYTES = spaceinBYTES + sysalloc * bytesPerCylinder
       End
END
"FREE FILE(file2)"
If spaceinBYTES > 0 then
Total  = spaceinBYTES / (1024 * 1024)
else Total = 0
queue centre('',80,'-')
queue centre('Summary of the space Used',80,'.')
queue centre('',80,'-')
queue centre('Total space used by above datasets = ',40),
      left(Total,10) left('Megabytes',15)
queue centre('Total files in this report         = ',40) left(IN.0,10)
queue centre('',80,'-')
return
Write_the_report:
/*-------------------------------------------------------------*/
/* Write the report                                            */
/*-------------------------------------------------------------*/
 IF queued() > 0 then do
       address tso
       outdsn = userid()||".test.report.dssize"
       IF sysdsn("'"outdsn"'")= 'OK' then
          do
            "delete '"||outdsn ||"'"
          end
  "alloc dd(report) ds('"outdsn"') recfm(f) dsorg(ps),
  lrecl(80) space(40,30) tracks new reu"
  IF rc > 0 then exit 8
    "execio "queued()" diskw report ( finis"
/*IF rc > 0 then exit 8 */
  "Ispexec browse dataset('"outdsn"')"
  "FREE FILE(report)"
 End
```

REPORT1

```
)ATTR
# TYPE(INPUT) INTENS(HIGH) CAPS(ON) COLOR(TURQUOISE) PAD('_') JUST(LEFT)
_ TYPE(INPUT) INTENS(HIGH) CAPS(ON) COLOR(GREEN)     PAD('_')
@ TYPE(TEXT)  INTENS(HIGH) CAPS(ON) COLOR(PINK)
! TYPE(TEXT)  INTENS(HIGH) CAPS(ON) COLOR(YELLOW)             SKIP(ON)
[ TYPE(TEXT)  INTENS(LOW)          COLOR(RED)                SKIP(ON)
{ TYPE(TEXT)  INTENS(HIGH)         COLOR(BLUE)               SKIP(ON)
} TYPE(TEXT)  INTENS(HIGH)         COLOR(GREEN)              SKIP(ON)
^ TYPE(TEXT)  INTENS(LOW)
)BODY
}%Userid :%&ZUSER  @REPORT :: DSN-UNIT-SPACE-ORG-LRECL-RECFM }%Date
```

```
:%&ZDATE
@                    ------   -----------------------------
%Command@===>_ZCMD
^
%-------------------------------------------------------------------------
--------
%Option @===>#y^
^
[1.!Report of Space Occupied by Particular Family of Datasets
   }==>#op1                                        [(Enter Partial
Qualifier)
^
} Enter User Id to see the DASD space occupied
^
[2.!Report of Space Occupied by Distinct Set of Datasets
   }==>#op2                                            [(Enter Dataset
Name)
^
} Do You want to recall the[Migrated Datasets}for creating the reports
  1 and 2 ?}==>#v!(Y/Blank)
  Use this option if you need LRECL & RECFM of Migrated Datasets.
@([CAUTION:@Recalling Migrated Dataset will take more execution time.)
^
%-------------------------------------------------------------------------
--------
{                          PF3 - Exit ; Enter - Process
%-------------------------------------------------------------------------
--------
)INIT
 Vget (keypress) PROFILE
&ZCMD = ' '
)PROC
 &KEYPRESS = .PFKEY
 VER(&y,NB)
 VER(&y,NUM)
 VPUT (y op1 op2 v
       Keypress) PROFILE
)END
```

Yash Pal Samnani
Program Analyst
Infosys Technologies Limited (USA)

Where are MIGrated datasets?

Using the WMIG CLIST you can easily obtain:

1 A scrollable list of all DFHSM migration volumes (panel WMI2).

2 A list of all migrated datasets on a volume (if you don't provide a LEVEL), or a subset of them (by supplying a LEVEL) (panel WMIGPAN).

3 A list of all migrated datasets on all HSM volumes, by supplying a LEVEL (panel WMIGPAN).

You can obtain HSM type (L1 or L2) for every volume, and, for all tape volumes, how full they are (FULL, PARTIAL, or EMPTY).

Select the search parameters in panel WMI2 as below:

```
Enter one or both of the following parameters:
  DSNAME LEVEL  ===> DSNA.A*
  VOLUME        ===> MIG003
```

You cannot use the % (per cent sign) character in the LEVEL field.

In the example, you obtain a list of all datasets prefixed by DSNA.A (the asterisk is optional) migrated on volume MIG003.

You may also select the volume by typing an 'S' in front of any volser in the list.

You may override the VOLUME selection by choosing one out of 138 HSM migration volumes listed below:

```
  LEVEL  VOLSER INFOS
  .....  ...... ................................
_ LEV 1  MIG000 THRESHOLD 080 ***
_ LEV 1  MIG001 THRESHOLD 080 ***
_ LEV 1  MIG002 THRESHOLD 080 ***
s LEV 1  MIG003 THRESHOLD 080 ***
_ LEV 1  MIG004 THRESHOLD 080 ***
_ LEV 1  MIG005 THRESHOLD 080 ***
_ L2-TP  HSM001 FULL
```

When you arrive in the second panel, WMIGPAN, you may select between various options:

1 L – list catalog information for the dataset.

2 R – recall dataset from tape or ML1 DASD.

3 M – migrate dataset from L1 to L2.

4 C or D – delete a migrated dataset.

5 S – select 3.4 panel.

6 Sorting information in the SORT CRITERIA field.

The information provided is shown below:

```
               Level ==> DSNA.A                          Row 1 to 5 of 5
COMMAND ===>                                           SCROLL ===> HALF

Dataset on ML1  ==> 5  Original TRKS ==> 8042     MB ==>    00000189
Dataset on ML2  ==> 0  Original TRKS ==> 0000             (ML1 plus ML2)
Total migrated  ==> 5  Tot.  MIGTRKS ==> 8042  compression factor 100:50

           SORT CRITERIA (1-6,a/d): 1A
   1                                  2      3        4        5      6
 S DSNAME                             MIGVOL LASTREFD MIGRDATE O.TRK OR
 . ................................. ...... ........ ........ ..... ..
   DSNA.ARCHLOG1.A0002387             MIG003 04/01/09 04/01/09 04000 PS
   DSNA.ARCHLOG1.A0002390             MIG003 04/01/09 04/01/09 04000 PS
   DSNA.ARCHLOG1.B0002387             MIG003 04/01/09 04/01/09 00014 PS
   DSNA.ARCHLOG1.B0002390             MIG003 04/01/09 04/01/09 00014 PS
   DSNA.ARCHLOG2.B0002387             MIG003 04/01/09 04/01/09 00014 PS
 ************************** Bottom of data ***************************
```

In this case, the five datasets found are all on disk (ML1); the 8042 original tracks have been compressed on ML1 with a factor 2:1 (100:50). Your datasets are always displayed (by default) sorted by dataset name, but you can sort them as you wish by entering a different sort criterion, in ascending or descending mode. For example, 5d means sort by original tracks, in descending order.

Warning: the WMIG CLIST recalls the panel WMI2 only if DFSMShsm is running.

Your userid must have DATABASEAUTHORITY(CONTROL) on

DFSMShsm member ARCCMD*xx* in order to issue HSEND commands inside the CLIST.

It works fine up to OS/390 2.10. I haven't tried it in z/OS. If you discover any problems please send an e-mail to a.mungai@tiscali.it.

In order to use the following members, put the CLIST member in a SYSPROC concatenated library (member=WMIG), eg ISP.UISPCLIB; put the two MESSAGE members in an ISPMLIB concatenated library (members=WMIG00 and WMIG01), eg ISP.UISPMLIB. Put the four PANEL members in an ISPPLIB concatenated library (member=WMI2, WMIGH, WMIGPAN, WMIGPANH), eg ISP.UISPCLIB.

WMIG CLIST

```
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* . .WMIG Clist . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
PROC Ø DEBUG
/*- SET UP FOR DEBUG IF REQUESTED ----------------------------------*/
  CONTROL NOMSG NOLIST NOFLUSH END(ENDO) NOCONLIST NOPROMPT
  IF &DEBUG  =  DEBUG THEN +
   CONTROL MSG LIST NOFLUSH END(ENDO) PROMPT SYMLIST CONLIST
/*- END OF SET UP ---------------------------------------------------*/
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* WMIG: TO LOOK FOR MIGRATED DATASETS AND PERFORM OPERATIONS          */
/*       AGAINST THEM.                                                 */
/*                                                                     */
/* RETURN CODES: Ø > PROCESSING COMPLETE                               */
/*              16 > UNEXPECTED ERROR IN DFHSM HSEND FUNCTION          */
/*              2Ø > DFSMSHSM NOT ACTIVE IN THIS SIDE                  */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
IF &SYSHSM= THEN DO
                 ISPEXEC SETMSG MSG(WMIGØ14)
                 EXIT CODE(2Ø)
               ENDO
SET &ARCI=ARCØ138I
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* EXTRACTING MIGRATION VOLUMES...                                     */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
  WRITE PLEASE WAIT, WE'RE LOADING MIGRATION VOLUMES LIST...
  DELETE '&SYSUID..WMIG'
  HSEND WAIT LIST MVOL ODS('&SYSUID..WMIG')
   ALLOC F(MIGVOLS) DA(WMIG) SHR REU
```

```
   OPENFILE MIGVOLS
   ISPEXEC TBCREATE W&SYSUID REPLACE SHARE +
   KEYS(TYPE MIGVOL DATI)
 ERROR DO
  IF &LASTCC = 4ØØ THEN GOTO FINE
  RETURN
     ENDO
 LEGG: +
   GETFILE MIGVOLS
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* EXTRACTING ML1 / ML2 SHORT INFOS TO FIT TABLE...                      */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
   SET &L = &SYSINDEX(LEV,&MIGVOLS)
   IF &L NE Ø THEN DO
   SET MIGVOL=&STR(&SUBSTR(1:17,&MIGVOLS))
   SET TYPE=&STR(&SUBSTR(18:22,&MIGVOLS))
   SET DATI=&STR(THRESHOLD &STR(&SUBSTR(26:32,&MIGVOLS)))
                ENDO
   ELSE DO
   SET &L = &SYSINDEX(L2-,&MIGVOLS)
   IF &L EQ Ø THEN GOTO LEGG
   SET MIGVOL=&STR(&SUBSTR(1:1Ø,&MIGVOLS)&SUBSTR(12:19,&MIGVOLS))
   SET TYPE=&STR(&SUBSTR(2Ø:24,&MIGVOLS))
   SET FULL=&STR(&SUBSTR(29:31,&MIGVOLS))
   SET EMPT=&STR(&SUBSTR(49:51,&MIGVOLS))
   IF &STR(&FULL)=YES AND &STR(&EMPT) = NO THEN +
                    SET     &DATI=FULL
   IF &STR(&FULL)=NO AND &STR(&EMPT) = YES THEN +
                    SET     &DATI=EMPTY
   IF &STR(&FULL)=NO AND &STR(&EMPT) = NO  THEN +
                    SET     &DATI=PARTIAL
      ENDO
   ISPEXEC TBADD W&SYSUID
   SET &NU=&NU+1
            GOTO LEGG
 FINE: +
   ERROR OFF
   CLOSFILE MIGVOLS
   ISPEXEC TBSORT W&SYSUID FIELDS(TYPE C A MIGVOL C A)
   ISPEXEC TBTOP W&SYSUID
WAI: +
SET &S=
SET &O1=Ø
SET &O2=Ø
SET &DL1=Ø
SET &DL2=Ø
SET &CRIT=&STR(DSNAME C A)
   SET ZS=1A
   SET &PRECZS = &ZS
   SET ZCMD=
```

```
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* DISPLAY RESULTS IN PANEL                                               */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
   ISPEXEC TBDISPL W&SYSUID PANEL(WMI2) +
       MSG(&MSG) CURSOR(&CSR) CSRROW(&CSRR) POSITION(CRP) AUTOSEL(NO)
   SET LQSTCC = &LASTCC
   IF &LQSTCC = 4 THEN DO
                ISPEXEC SETMSG MSG(WMIGØ13)
                GOTO WAI
                     ENDO
   IF &LQSTCC = 8 THEN GOTO ESCI
   IF &LQSTCC = Ø AND &S NE &STR( ) THEN DO
                SET WVOLUM=&MIGVOL
                ISPEXEC VPUT WVOLUM
                                          ENDO
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* EXTRACTING LEVEL...                                                    */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
ISPEXEC VGET WDSN
SET &LEVEL=&STR(&WDSN)
  IF &STR(&LEVEL)=  THEN SET &LEVEL=DSN
  ELSE DO
  SET &L  = &SYSINDEX(%,&LEVEL)
  IF &L GT Ø THEN DO
                 ISPEXEC SETMSG MSG(WMIGØ19)
                 GOTO WAI
                ENDO
  SET &L  = &SYSINDEX(*,&LEVEL)
  IF &L GT Ø THEN DO
                 SET &L = &L-1
                 SET &LEVEL = &STR(&SUBSTR(1:&L,&LEVEL))
                 SET &M = Ø
             SOP: +
                 SET &L  = &SYSINDEX(.,&LEVEL,&M)
                 IF &L GT 8 THEN DO
                 ISPEXEC SETMSG MSG(ISRU187)
                 GOTO WAI
                              ENDO
                 SET &L  = &SYSINDEX( ,&LEVEL,&M)
                 IF &L GT 8 THEN DO
                 ISPEXEC SETMSG MSG(ISRU187)
                 GOTO WAI
                              ENDO
                 IF &L NE Ø THEN DO
                 SET &M = &L
                 GOTO SOP
                              ENDO
                ENDO
  SET &LEVEL = &STR(LEVEL(&LEVEL))
     ENDO
```

```
 ISPEXEC VGET WVOLUM
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* CHECK IF VOLSER WAS SUPPLIED                                          */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
   IF &STR(&WVOLUM) = MIGRAT OR &STR(&WVOLUM) =  THEN SET &SEL=
   ELSE DO
   SET LLL=&LENGTH(&STR(&WVOLUM))
   IF &LLL < 6 THEN DO
   ISPEXEC SETMSG MSG(WMIGØØ7)
   GOTO WAI
                  ENDO
   IF &LLL > 6 THEN SET &LLL=6
   SET WVOLUM=&STR(&SUBSTR(1:&LLL,&WVOLUM))
   SET &SEL=&STR(SEL(VOL(&WVOLUM)))
      ENDO
 DELETE '&SYSUID..OUTLML'
 WRITE PLEASE WAIT, WE'RE PERFORMING YOUR REQUEST:
 WRITE HSEND WAIT LIST &LEVEL MCDS &SEL ODS(&SYSUID..OUTLML)
 HSEND WAIT LIST &LEVEL MCDS &SEL ODS(&SYSUID..OUTLML)
 IF &LASTCC NE Ø THEN DO
                 ISPEXEC SETMSG MSG(WMIGØ18)
 EXIT CODE(16)
                    ENDO
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* CREATING MIGTAB TABLE...                                              */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
    ISPEXEC TBCREATE M&SYSUID REPLACE SHARE +
    KEYS(DSNAME) NAMES(VOL LASTREF LY MIGDATE MY ORIGTRK DSORG)
 ALLOC F(INP) DA('&SYSUID..OUTLML') SHR REU
 OPENFILE INP INPUT
 ERROR +
 DO
  IF &LASTCC = 8   THEN GOTO ESCI
  IF &LASTCC = 4ØØ THEN GOTO CHIUDI
  RETURN
 ENDO
 LEGGI: +
    GETFILE  INP
    SET &E = &SYSINDEX(&ARCI,&INP)
 IF &E NE Ø THEN GOTO CHIUDI
    SET MITRA=&SUBSTR(71:78,&INP)
    SET BYTES=&SUBSTR(73:77,&INP)
 IF &STR(&BYTES)=&STR(BYTES) THEN SET &B=&SUBSTR(71:71,&INP)
 IF &DATATYPE(&STR(&MITRA)) = NUM THEN SET MEGAB=&STR(&MITRA)
    SET CONTR1=&SUBSTR(68:68,&INP)
    SET CONTR2=&SUBSTR(71:71,&INP)
 IF &STR(&CONTR1) NE &STR(/) OR &STR(&CONTR2) NE &STR(/) THEN +
    GOTO LEGGI
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* LOADING DATA INTO MIGTAB TABLE...                                     */
```

```
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
 CARICA: +
    SET &VOL=&SUBSTR(49:54,&INP)
    SET &DSNAME=&SUBSTR(2:45,&INP)
    SET &DSNAME=&DSNAME
    SET &LASTREF=&SUBSTR(57:64,&INP)
        SET &Y2KL=&SUBSTR(57:58,&INP)
         IF &Y2KL < 5Ø THEN SET &LY=&STR(Ø&LASTREF)
         ELSE SET &LY=&STR(.&LASTREF)
    SET &MIGDATE=&SUBSTR(66:73,&INP)
        SET &Y2KM=&SUBSTR(66:67,&INP)
         IF &Y2KM < 5Ø THEN SET &MY=&STR(Ø&MIGDATE)
         ELSE SET &MY=&STR(.&MIGDATE)
    SET &DSORG=&SUBSTR(96:97,&INP)
    SET &ORIGTRK=&SUBSTR(76:8Ø,&INP)
    SET &LEVELLO=&SUBSTR(82:88,&INP)
    IF &STR(&LEVELLO) NE &STR(*******) THEN DO
    SET DL1=&DL1+1
    SET O1=&O1+&ORIGTRK
                                    ENDO
    ELSE DO
    SET DL2=&DL2+1
    SET O2=&O2+&ORIGTRK
       ENDO
    ISPEXEC TBADD M&SYSUID MULT(1ØØØ)
    GOTO LEGGI
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* DISPLAY TABLE AND SEARCH RESULTS                                        */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
 CHIUDI: +
   ERROR OFF
   IF &DL1 = Ø AND &DL2 = Ø THEN DO
      CLOSFILE INP
      IF &E NE Ø THEN +
      ISPEXEC SETMSG MSG(WMIGØØ9)
      ELSE +
      ISPEXEC SETMSG MSG(WMIGØØ5)
      GOTO WAI
                                ENDO
   SET TO=&O1+&O2
   IF &B = K THEN SET &MIGAB=&MEGAB*1ØØ
   IF &B = M THEN SET &MIGAB=&MEGAB*1ØØØØØ
   IF &B = G THEN SET &MIGAB=&MEGAB*1ØØØØØØØØ
   SET CPR=&MIGAB/(&TO*47)
   SET &LL=&LENGTH(&TO)
   SET &L1=&LENGTH(&O1)
   SET &L2=&LENGTH(&O2)
   IF &LL > &L1 THEN DO
      DO UNTIL &LL = &L1
         SET &O1=&STR(Ø&O1)
```

```
         SET &L1=&L1+1
    ENDO
                   ENDO
   IF &LL > &L2 THEN DO
      DO UNTIL &LL = &L2
         SET &O2=&STR(Ø&O2)
         SET &L2=&L2+1
    ENDO
                   ENDO
   SET TL=&DL1+&DL2
   SET &LL=&LENGTH(&TL)
   SET &L1=&LENGTH(&DL1)
   SET &L2=&LENGTH(&DL2)
   IF &LL > &L1 THEN DO
      DO UNTIL &LL = &L1
         SET &DL1=&STR(Ø&DL1)
         SET &L1=&L1+1
    ENDO
                   ENDO
   IF &LL > &L2 THEN DO
      DO UNTIL &LL = &L2
         SET &DL2=&STR(Ø&DL2)
         SET &L2=&L2+1
    ENDO
                   ENDO
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* SORTING TABLE...                                                        */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
CHIUD1: +
   ISPEXEC TBSORT M&SYSUID FIELDS(&CRIT)
   IF &LENGTH(&LASTREF) > 8 THEN DO
      SET &LASTREF=&STR(&SUBSTR(3:1Ø,&LASTREF))
                               ENDO
      ISPEXEC TBTOP M&SYSUID
 SET CSR =
 SET CSRR=1
 SET MSG =
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* DISPLAY WMIGPAN PANEL                                                   */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
 TBD: +
      ISPEXEC TBDISPL M&SYSUID PANEL(WMIGPAN) +
         MSG(&MSG) CURSOR(&CSR) CSRROW(&CSRR) POSITION(CRP) AUTOSEL(NO)
      SET &RC = &LASTCC
      IF &RC = 4 THEN DO
          ISPEXEC SETMSG MSG(WMIGØØ1)
          GOTO TBD
                   ENDO
TRC:  IF &RC = Ø AND &SCELTA EQ &STR( ) AND &PRECZS = &ZS THEN DO
                   ISPEXEC SETMSG MSG(WMIGØØ8)
```

```
                  SET CSRR = &CRP
                  ISPEXEC TBDISPL M&SYSUID POSITION(CRP)
                  SET &RC = &LASTCC
                  GOTO TRC
                                                              ENDO
      IF &RC = 8 THEN DO
                 CLOSFILE INP
                 ISPEXEC TBCLOSE M&SYSUID
                 GOTO WAI
                    ENDO
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* REDIRECT SELECTION...                                                 */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
      IF &RC = Ø THEN IF &ZTDSELS = 1 AND &SCELTA NE &STR( ) THEN DO
                   IF &SCELTA EQ &STR(L) THEN GOTO LC
                   IF &SCELTA EQ &STR(R) THEN GOTO REC
                   IF &SCELTA EQ &STR(M) THEN GOTO MIG2
 IF &SCELTA = &STR(D) OR &SCELTA EQ &STR(C) THEN GOTO CANC
             GOTO TRE4
                                                                 ENDO
 POI34: IF &ZTDSELS GT 1 THEN DO
                   IF &SCELTA EQ &STR( ) THEN GOTO TRC
                   IF &SCELTA EQ &STR(L) THEN GOTO LC
                   IF &SCELTA EQ &STR(R) THEN GOTO REC
                   IF &SCELTA EQ &STR(M) THEN GOTO MIG2
 IF &SCELTA = &STR(D) OR &SCELTA EQ &STR(C) THEN GOTO CANC
                   ISPEXEC CONTROL DISPLAY SAVE
                   GOTO TRE4
 POI34GT:          ISPEXEC CONTROL DISPLAY RESTORE
                   GOTO TBD
                            ENDO
     SET CSRR = &CRP
     ISPEXEC VGET ZS
     IF &ZS = &PRECZS THEN GOTO TBD
     SET &PRECZS = &ZS
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* SORT OPTIONS                                                          */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
     SELECT &ZS
       WHEN (1A) SET &CRIT=&STR(DSNAME C A)
       WHEN (2A) SET &CRIT=&STR(VOL C A)
       WHEN (3A) SET &CRIT=&STR(LY C A)
       WHEN (4A) SET &CRIT=&STR(MY C A)
       WHEN (5A) SET &CRIT=&STR(ORIGTRK C A)
       WHEN (6A) SET &CRIT=&STR(DSORG C A)
       WHEN (1D) SET &CRIT=&STR(DSNAME C D)
       WHEN (2D) SET &CRIT=&STR(VOL C D)
       WHEN (3D) SET &CRIT=&STR(LY C D)
       WHEN (4D) SET &CRIT=&STR(MY C D)
       WHEN (5D) SET &CRIT=&STR(ORIGTRK C D)
```

```
       WHEN (6D) SET &CRIT=&STR(DSORG C D)
     OTHERWISE GOTO TBD
               ENDO
      GOTO CHIUD1
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* CANC SUBROUTINE: C OR D, DELETE MIGRATED DATASET                      */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
CANC: +
ISPEXEC ADDPOP ROW(5) COLUMN(1)
ISPEXEC DISPLAY PANEL(MEMOPOP1)
IF &LASTCC = Ø THEN DO
           HSEND WAIT DELETE &DSNAME PURGE
           IF &LASTCC NE Ø THEN +
             WRITE  &DSNAME DELETE NOT PROCESSED
           ELSE +
           SET &VOL=§§§DEL
           ISPEXEC TBMOD M&SYSUID ORDER
           SET &CMD=DELETE
           IF &ZTDSELS EQ 1 THEN ISPEXEC SETMSG MSG(WMIGØØ2)
                  ENDO
           ISPEXEC REMPOP
           IF &ZTDSELS NE 1 THEN GOTO TBD
           GOTO POI34
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* REC  SUBROUTINE: R, RECALL ON DISK A MIGRATED DATASET                 */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
REC: +
           SET &VOL=§§§REC
ISPEXEC TBMOD M&SYSUID ORDER
           HRECALL '&DSNAME'
           SET &CMD=RECALL
           ISPEXEC SETMSG MSG(WMIGØØ2)
           IF &ZTDSELS = 1 THEN GOTO POI34
           ELSE GOTO TBD
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* MIG2 SUBROUTINE: M, MIGRATE AN ML1 DATASET TO ML2                     */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
MIG2: +
           SET &VO2=&SUBSTR(1:3,&VOL)
           IF &VO2=HSM THEN DO
             ISPEXEC SETMSG MSG(WMIGØØ4)
             GOTO AV2
                         ENDO
           SET &VOL=§§§ML2
ISPEXEC TBMOD M&SYSUID ORDER
           HMIG    '&DSNAME'  ML2
           ISPEXEC SETMSG MSG(WMIGØØ3)
 AV2:      IF &ZTDSELS = 1 THEN GOTO POI34
           ELSE GOTO TBD
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
```

```
/* 3.4  SUBROUTINE: S, HOOK UP TO 3.4 ISPF PANEL                    */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
TRE4: +
             ISPEXEC CONTROL NONDISPL ENTER
             ISPEXEC CONTROL DISPLAY LOCK
             SET &ZDLDSAVE = &STR(&ZDLDSNLV)
             SET &ZDLDSNLV = &DSNAME
             SET &ZDLPVL   =
              ISPEXEC VPUT (ZDLDSNLV ZDLPVL)
             ISPEXEC SELECT PGM(ISRUDL) PARM(WULL)
             SET &ZDLDSNLV = &STR(&ZDLDSAVE)
             ISPEXEC VPUT ( ZDLDSNLV)
DOP4:      IF &ZTDSELS = 1 AND &SCELTA NE &STR( )  THEN GOTO POI34
           ELSE GOTO POI34GT
             GOTO WAI
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* LC  SUBROUTINE: L, LISTCAT DATASET ENTRIES TO CONTROL CATALOG    */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
LC: +
ALLOC F(TEMP) DA('&SYSUID..LISTCAT') SHR REU
IF &LASTCC NE Ø THEN +
 ALLOC F(TEMP) DA('&SYSUID..LISTCAT') NEW REU RECFM(V B A) LRECL(125)
LISTC ENT('&DSNAME') ALL  OUTFILE(TEMP)
 SET &RC = &LASTCC
   IF &RC = Ø THEN  ISPEXEC BROWSE DATASET('&SYSUID..LISTCAT')
   IF &RC = 4 THEN  ISPEXEC SETMSG MSG(WMIGØ16)
   IF &RC > 4 THEN  ISPEXEC SETMSG MSG(WMIGØ17)
 GOTO TBD
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* END OF JOB                                                       */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
 ESCI: +
 ISPEXEC TBCLOSE W&SYSUID
 ISPEXEC SETMSG MSG(WMIGØØ6)
 FREE F(INP)
 EXIT CODE(Ø)
```

WMIG00 MESSAGES

```
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* . .WMIGØØ Messages. . . . . . . . . . . . . . . . . . . . . . . */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
WMIGØØ1 'NOT ALLOWED MULTISELECTION'  .ALARM=YES
'Enter a single option, multiple selection not allowed'
WMIGØØ2 '&CMD SUBMITTED'   .ALARM=NO
'&CMD submitted for &DSNAME'
WMIGØØ3 'MIGRATION 1>2 SUBMITTED' .ALARM=NO
'HMIG ML2 accepted for &DSNAME'
WMIGØØ4 '&VOL MIGRATION NOT ALLOWED' .ALARM=YES
```

```
'Migration not allowed between ML2 media'
WMIG005 'NO DATASET MIGRATED'  .ALARM=YES
'No migrated dataset can be found according to your request'
WMIG006 'WMIG ENDED'   .ALARM=NO
'Clist WMIG completed'
WMIG007 '&WVOLUM INVALID VOLSER' .ALARM=YES
'Please enter a valid volume name'
WMIG008 'PLEASE MAKE A SELECTION' .ALARM=YES
'To get an help selection list, press PF1'
WMIG009 '&WVOLUM NO DATASET FOUND' .ALARM=YES
'No MCDS information found for volume &WVOLUM'
```

WMIG01 MESSAGES

```
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* . .WMIG01 Messages  . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
WMIG010 'LEVEL SYNTAX ERROR'  .ALARM=YES
'Supply a level (followed by an *) or a fully qualified dataset name'
WMIG011 'VOLSER SYNTAX ERROR'  .ALARM=YES
'Supply a valid 6-char volume name, or select it in the list below'
WMIG012 'ENTER A VALUE'  .ALARM=YES
'Supply a level, a dsname or a volume according the list below'
WMIG013 'TOO MANY SELECTIONS'  .ALARM=YES
'Make only ONE selection left to volser in the list'
WMIG014 'DFSMShsm INACTIVE'  .ALARM=YES
'DFSMShsm is not active, WMIG needs HSM to perform its search'
WMIG015 'SELECTION ERROR'  .ALARM=YES
'Make a selection between S, C/D, M, R (PF1 to read instructions)'
WMIG016  'ENTRY NOT LISTED'   .ALARM=YES
'MSG IDC0014I, LASTCC = &RC'
WMIG017  'LISTCAT FAILED, RC=&RC'   .ALARM=YES
'LISTCAT ERROR, RC = &RC'
WMIG018  'DFSMShsm ERROR'   .ALARM=YES
'DFSMShsm returns an unexpected error. Control DFHSM output'
WMIG019   '% CHAR FORBIDDEN' .ALARM=YES
'You cannot use % char in the LEVEL field, please correct'
```

WMI2 PANEL

```
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* . . WMI2 Panel  . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
)ATTR DEFAULT(%+_)
 § TYPE(input) COLOR(GREEN) pad(_)
 [ TYPE(TEXT) COLOR(GREEN)
 $ TYPE(output) COLOR(PINK)
 £ TYPE(output) COLOR(TURQ)
```

```
 ] TYPE(TEXT) COLOR(YELLOW)
)BODY
]----------------- SEARCH FOR MIGRATED DATASETS ----------------------
--------
%OPTION  ===>_ZCMD
+
%
+Enter one or both of the following parameters:
]  DSNAME LEVEL %===>_WDSN                                          +
]  VOLUME       %===>_WVOLUM+
%
% You may override VOLUME selection, by choosing one out of
] &NU%HSM Migration Volumes listed below:
+   LEVEL  VOLSER INFOS
+   .....  ...... ............................
)MODEL ROWS(ALL)
 §S£TYPE $MIGVOL £DATI
)INIT
  vget (wdsn wvolum) profile
  .help = wmigh
)REINIT
  &s = ' '
  .AUTOSEL = NO
)PROC
    IF (&ZTDSELS = '0000')
           IF (&wvolum = ' ')
             VER (&wdsn,nonblank,MSG=WMIG012)
             VER (&wdsn,dsnamef,MSG=WMIG010)
           else
             VER (&wvolum,pict,CCCCCC,MSG=WMIG011)
             VER (&wdsn,dsnamef,MSG=WMIG010)
         IF (&wdsn = , ,)
           VER (&wvolum,nonblank,MSG=WMIG012)
           VER (&wvolum,pict,CCCCCC,MSG=WMIG011)
    IF (&ZTDSELS = '0001')
         IF (&WDSN NE ' ')
           VER (&wdsn,dsnamef,MSG=WMIG010)
  vput (wdsn wvolum) profile
)END
```

WMIGH HELP PANEL

```
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* . . WMIGH Help Panel  . . . . . . . . . . . . . . . . . . . . . . */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
)ATTR DEFAULT(%+_)
 [ TYPE(TEXT) COLOR(GREEN)
 ^ TYPE(TEXT) COLOR(red)
 $ TYPE(TEXT) COLOR(PINK) hilite(blink)
```

```
 ] TYPE(TEXT) COLOR(TURQ)
 £ TYPE(TEXT) COLOR(YELLOW)
)BODY
%HELP                          £SEARCH FOR MIGRATED DATASETS
+
  You may search for:
 ]a.+all datasets on a single HSM volume (tape or disk) supplying VOLUME
     on the VOLUME line or selecting one volser out of the listed
     HSM Migration Volumes. The list is scrollable.
 ]b.+migrated datesets providing a LEVEL
 ]c.+a combination of]a.+and]b.+
      £eg.:
      +DSNAME LEVEL %===>^db2.a*]<-- search for all 'db2.a*' migrated
dataset
      +VOLUME       %===>^HSMØØØ]<-- limit search just to this volume

 $WARNING:+
  The£WMIG+clist recall the panel£WMI2]only if DFSMShsm is running.
 +Your userid must have]DATABASEAUTHORITY(CONTROL) on DFSMShsm ARCCMDxx
 +in order to issue HSEND commands inside the clist.

  Press£ENTER+or£PF3+to return.
)INIT
)PROC
  .RESP = END
)END
```

WMIGPAN PANEL

```
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* . .WMIGPAN Panel  . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
)ATTR DEFAULT(£+§)
 [ TYPE(OUTPUT) COLOR(YELLOW)
 # TYPE(text) COLOR(YELLOW)
 ! TYPE(OUTPUT) COLOR(WHITE)
 £ TYPE(text) COLOR(WHITE)
 & TYPE(TEXT) COLOR(blue)
 $ TYPE(TEXT) COLOR(green)
 ] TYPE(TEXT) COLOR(TURQ)
 ^ TYPE(OUTPUT) INTENS(LOW) PAD(' ') JUST(left)
)BODY
$                       Level ==>]&WDSN
+ COMMAND ===>§ZCMD                                    &SCROLL ===>§AMT
+
+
] Dataset on ML1  ==>#&DL1  ]Original TRKS ==>#&O1     ]&B.B ==> #&MEGAB
] Dataset on ML2  ==>#&DL2  ]Original TRKS ==>#&O2              +(ML1
plus ML2)
```

```
] Total migrated  ==>#&TL   ]Tot.  MIGTRKS ==>#&TO  ]compression
factor$100:&CPR
+
+          SORT CRITERIA (1-6,a/d):§ZS+
+
  +1                                          2      3        4
5      6
]S DSNAME                                     MIGVOL LASTREFD MIGRDATE
O.TRK OR
]. ........................................ ...... ........ ........
..... ..
)MODEL ROWS(ALL)
§Z!Z                                         [Z     !Z       ^Z
^Z    ^Z
)INIT
  .HELP  = WMIGPANH
  .ZVARS = '(SCELTA DSNAME VOL LASTREF MIGDATE ORIGTRK DSORG)'
  .CURSOR = ZS
  &AMT = HALF
  &SCELTA= , ,
)REINIT
 IF (.MSG = ' ')
  &SCELTA= ' '
  &AMT = HALF
  REFRESH(SCELTA)
)PROC
   VER(&ZS,NB,LIST,1A,1D,2A,2D,3A,3D,4A,4D,5A,5D,6A,6D)
       IF (.RESP = ENTER)
           vput (zs)
  IF (&ZTDSELS NE 0000)
   VER (&SCELTA,LIST,l,L,s,S,c,C,d,D,r,R,m,M,MSG=WMIG015)
)END
```

WMIGPANH HELP PANEL

```
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* . .WMIGPANH Help Panel  . . . . . . . . . . . . . . . . . . . . */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
)ATTR DEFAULT(%+_)
 [ TYPE(TEXT) COLOR(GREEN)
 $ TYPE(TEXT) COLOR(PINK)
 ] TYPE(TEXT) COLOR(TURQ)
 £ TYPE(TEXT) COLOR(YELLOW)
)BODY
%HELP         £SELECTION FROM MIGRATED DATASET LIST
+Page 1/1
+
+   The panel provides the following information, according to the
supplied
```

```
+ level and volume:
  + N. of dataset on[ML1+and original tracks allocation
  + N. of dataset on[ML2+and original tracks allocation
  + Total datasets and total migrated tracks
  + Total K/M/Gbytes migrated
+
+  Select£SORT CRITERIA (1-6,a/d)+; you may sort data by chosen column +
+     eg: ]1d+means]sort by dsname, descending                          +
+         ]2a+means]sort by volser, ascending                           +
+
+   Column£S+lets you make a line selection. Enter:
+
 ]S+- to display£DSLIST (3.4)+panel for the selected dataset            +
 ]L+- to do a£LISTCAT ALL+without recalling the selected dataset        +
 ]M+- to£migrate+a ML1 dataset to£ML2+                                  +
 ]R+- to£recall on disk+the selected dataset                            +
 ]C+or]D+-to£cancel+the selected dataset                                +
+
)INIT
)PROC
  .RESP = END
)END
```

MEMOPOP1 PANEL

```
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/* . MEMOPOP1 Panel  . . . . . . . . . . . . . . . . . . . . . . . . */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
)ATTR DEFAULT(%+_)
 [ TYPE(TEXT) COLOR(RED)
 ] TYPE(TEXT) COLOR(TURQ)
 £ TYPE(TEXT) COLOR(GREEN)
)BODY WINDOW(28,3)
£
£ENTER =]cancel £PF3]= no!
£
)END
```

Alberto Mungai
Senior Systems Programmer (Italy)

ESCON Director display utility

INTRODUCTION

ESCON Director, sometimes referred to as a ‘dynamic switch’, is a switch that acts as a communications hub for ESCON channels.

It provides the capability to physically interconnect any two links that are attached to it. Such a connection between two ports provides simultaneous two-way information transfer. When a connection is established, the two ports and their respective point-to-point links are connected so that frames received by one of the ports are passed transparently to the other port. Such a connection can be either static or dynamic.

ESCON Directors have a major role in MVS data centres. Systems programmers have to manage them in order to plan their hardware configurations.

Because they are located in computer rooms, it is often painful to get their active configuration from their console – you have to move!

Of course, you can use the MVS D M=SWITCH command to get detailed information about a specific ESCON Director port, but there is no standard utility to get a global view.

The utility described in this article is a REXX program that retrieves an ESCON Director configuration from D M=SWITCH commands and displays it on an ISPF panel.

ESCDCONF REXX

```
/* REXX                                                    */
/* REXX routine to display ESCD configuration              */
/*                                                         */
WAIT_TIME = 2                 /* wait time for console service */
s = p                         /* default sort key              */
"ISPEXEC TBCREATE ESCDCONF
```

```
         NAMES(PORT PORTD STATUS ND)
         REPLACE"
"CONSPROF SOLDISPLAY(NO) UNSOLDISPLAY(NO)",
         "SOLNUM(9999) UNSOLNUM(Ø)"
"CONSOLE ACTIVATE"
IF RC <> Ø THEN
  DO
   "CONSOLE DEACTIVATE"
   IF RC <> Ø THEN
    DO
     SAY '*** USERID' USERID() 'NEEDS CONSOLE AUTHORITY'
     EXIT 8
    END
  END
"ISPEXEC TBDISPL ESCDCONF PANEL(ESCDØØ) CURSOR(DEVN)"
DISPRC = RC
DO WHILE DISPRC = Ø             /* until PF3 */
 IF WORDS(ZCMD) >= 1 THEN       /* locate ? */
   DO
     CMD     = WORD(ZCMD,1)
     PORTX   = LEFT(WORD(ZCMD,2),2,'Ø')
     IF (PORTX > "FF" AND PORTX < "ØØ") THEN PORTX = "FF"
     PORTD = X2D(PORTX)
     SELECT
      WHEN CMD = L THEN
        DO
          "ISPEXEC TBTOP  ESCDCONF"
          "ISPEXEC TBSCAN ESCDCONF ARGLIST(PORTD) CONDLIST(GE)"
          "ISPEXEC TBDISPL ESCDCONF PANEL(ESCDØØ)"
          disprc = rc
        END
      OTHERWISE
        DO
        END
     END
   END
  ELSE
   DO
     "ISPEXEC TBCREATE ESCDCONF
              NAMES(PORT PORTD STATUS ND)
              REPLACE"
     P1X = SP
     P2X = EP
     ESCD_ADDR = DEVN
     sortkey = s
     /* CHECK ESCD DEVICE NUMBER */
     CMD = "D U,,,"ESCD_ADDR",1"
     CARTVAL = USERID()||TIME()
     "CONSOLE SYSCMD("CMD") CART("CARTVAL")"
     GET_RC = GETMSG('RESP.','SOL',CARTVAL,,WAIT_TIME)
```

```
      IF WORD(RESP.3,2) = "SWCH" THEN
        DO
         P1D = X2D(P1X)
         P2D = X2D(P2X)
         /* CHECK ESCD PORT RANGE */
         IF P1D > P2D THEN
           DO
            "ISPEXEC SETMSG MSG(ESCDØØ2E)"
           END
          ELSE
           DO
            DO ID = P1D TO P2D
             IX = D2X(ID)
             CMD = "D M=SWITCH("ESCD_ADDR","IX")"
             CARTVAL = USERID()||TIME()
             "CONSOLE SYSCMD("CMD") CART("CARTVAL")"
             GET_RC = GETMSG('RESP.','SOL',CARTVAL,,WAIT_TIME)
             PORT = RIGHT(IX,2,'Ø')
             PORTD = ID
             PARSE VAR RESP.2 "STATUS=" STATUS
             PARSE VAR RESP.3 "NODE = " ND
             "ISPEXEC TBADD ESCDCONF"
            END
            "ISPEXEC TBTOP   ESCDCONF"
           END
        END
       ELSE
        DO     /* it is not a ESCD devnum */
         "ISPEXEC SETMSG MSG(ESCDØØ1E)"
        END
      select
       when sortkey = "P" then sk = "PORTD"
       when sortkey = "S" then sk = "STATUS,c,A,portd,n,A"
       when sortkey = "N" then sk = "ND,c,A,portd,n,A"
      end
      "ISPEXEC TBsort  ESCDCONF fields("sk")"
      "ISPEXEC TBDISPL ESCDCONF PANEL(ESCDØØ) CURSOR(DEVN)"
      DISPRC = RC
      "ISPEXEC CONTROL DISPLAY SAVE"
    END
END
"CONSOLE DEACTIVATE"
```

ESCD01 ISPF PANEL

```
)ATTR
   ! TYPE(OUTPUT) INTENS(LOW) JUST(LEFT)
   ¢ TYPE(OUTPUT) INTENS(HIGH) JUST(LEFT)
   # TYPE(TEXT) COLOR(RED) INTENS(HIGH)
```

```
   \ TYPE(TEXT) COLOR(YELLOW) INTENS(HIGH)
   % TYPE(TEXT) COLOR(GREEN) INTENS(HIGH)
   $ TYPE(TEXT) SKIP(ON) INTENS(LOW)
)BODY EXPAND(@@)
+@-@#Escon Director Configuration+@-@+
$\COMMAND%===>_ZCMD                                          %SCROOL
===>_SAMT+
$
$+ESCD devnum%===>_devn+ Start port%===>_sp+ End port%===>_ep+Sort
key%===>_s+
$
$%Port+%DCM Status+               %Node Descriptor
$
)MODEL
$!Z   +¢Z                      +  !Z                                  +
)INIT
 .ZVARS = '(port status nd)'
 &ZTDMARK = '****************************** BOTTOM OF DATA +
******************+
*************'
)PROC
  IF (.RESP = ENTER)
   VER (&devn,NONBLANK)
   VER (&sp,NONBLANK,hex)
   VER (&ep,NONBLANK,hex)
   VER (&s,list,P,S,N)
)help field(devn) msg(escd001h)
      field(sp) msg(escd002h)
      field(ep) msg(escd002h)
      field(s)  msg(escd003h)
)END
```

SAMPLE DISPLAY

In order to use ESCDCONF, you need to have TSO CONSOLE authority.

```
------------------ Escon Director Configuration ------ Row 1 to 16 of 16
 COMMAND ===>                                           SCROLL ===> PAGE

ESCD devnum ===> 9001 Start port ===> D0 End port ===> DF Sort key ===>P

  Port  DCM Status                 Node Descriptor

  D0    NOT ATTACHED               UNKNOWN
  D1    CHANNEL ATTACHED           002064.2C5.IBM.51.000000069150
  D2    CHANNEL ATTACHED           002084.304.IBM.83.0000000292CA
  D3    NOT DCM ELIGIBLE           003490.A20.STK.10.000000008867
  D4    CHANNEL ATTACHED           002064.2C5.IBM.51.000000069150
```

```
  D5    NOT ATTACHED                UNKNOWN
  D6    NOT DCM ELIGIBLE            003490.C22.IBM.77.0000000D5350
  D7    NOT ATTACHED                UNKNOWN
  D8    CHANNEL ATTACHED            002064.2C5.IBM.51.000000069150
  D9    CHANNEL ATTACHED            002064.2C5.IBM.51.000000069150
  DA    NOT DCM ELIGIBLE            003746.900.IBM.57.000000092804
  DB    CHANNEL ATTACHED            002064.2C5.IBM.51.000000069150
  DC    OFFLINE, PORT ATTACHMENT    UNKNOWN
  DD    NOT DCM ELIGIBLE            003490.A10.STK.03.000000310712
  DE    CHANNEL ATTACHED            002084.304.IBM.83.0000000292CA
  DF    CHANNEL ATTACHED            002064.2C5.IBM.51.000000069150
************************** BOTTOM OF DATA ***************************
```

Alexandre Goupil
Systems Programmer (France)

Making the transition from Assembler to C on the mainframe

PROBLEM ADDRESSED

Many installations are faced with the problem of legacy Assembler code. There are several problems associated with this legacy code:

- The number of available Assembler programmers is rapidly declining.
- Assembler code, being 'near the metal', is susceptible to hardware changes. For example, the recent change to the cache architecture caused a significant performance degradation for some common programming techniques. Such effects, obviously, cannot be ruled out in the future. Similarly, the availability of 64-bit general purpose registers has introduced a new complexity to Assembler programming; the number of instructions has increased to 569. Although obviously not directly comparable, the C language has just 11 statements and 30 operators (some of which can be

combined). However, this gives a general indication of the problem of ‘modern’ Assembler programming.

These aspects can be mitigated by using a compiled high-level programming language. Although in many installations C is already a legacy language for PC application programming, it (and, more particularly, SPC – System Programming Facilities for C) is the only appropriate replacement for Assembler for low-level programming tasks on the mainframe, even though it cannot replace Assembler for all tasks. However, it often suffices to write critical functions in Assembler that can then be called from C; this allows the principal logic to be written in C. In a future article, I will describe an example of such a function.

My experience has shown that many traditional Assembler programmers have difficulty in making the transition to the C language constructs. Because almost no books discuss the special aspects of C mainframe programming, this article helps to ease this transition by relating the C language constructs to the mainframe Assembler equivalents (at least to the extent possible) and to highlight some potential problem areas. This article does not aim to provide an exhaustive description of the C programming language. Similarly, it does not discuss C++ aspects; this is because I feel Assembler programmers will first make the transition to C before, possibly, going on to more advanced object-oriented programming with C++.

C	*Assembler*	*Length*
char	C	1
short	H	2
int	F	4
long	F	4
long long	FD	8

Figure 1: C elementary data items and Assembler field types

ELEMENTARY DATA ITEMS

Figure 1 lists the common C elementary data items with the comparable Assembler field type. Observe that in z/OS C (the IBM C mainframe compiler), int and long fields are both 4 bytes long. z/OS C 1.5 provides 64-bit support with the long long field that is 8 bytes long. Note: this table is not exhaustive.

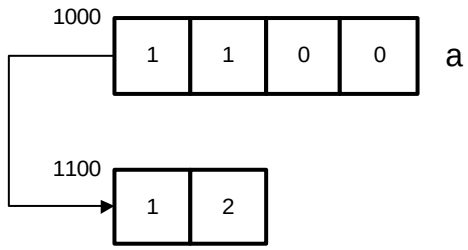
FIELD ASSIGNMENT

Field assignment, namely moving one field to another, is the most frequent operation in any programming language. This is illustrated in Figure 2.

Direct assignment

```
x = y;
```

This is the simplest assignment: the contents of <y> are assigned to <x>. Because C performs type checking, x and y must be comparable elementary data types. Under some circumstances y will be converted, for example, when x is a fullword (int) and y is a halfword (short).




```
b = a; /* 1100 */
b = &a; /* 1000 */
b = *a; /* 12 */
```

Figure 2: Field assignment

Assembler pseudo-code:

```
MVC  x,y
```

This is true only in the simplest case for exactly compatible types. Run-time functions (or coded loops) must be used to move non-elementary data types (aggregates), such as character strings, structures, etc.

Direct assignment with explicit type (casting)

```
x = (int)y; /* for example */
```

A cast, written within parentheses, can be used to specify that the result is to have a specific type. This can be used to avoid compiler warnings when the field types are incompatible. In the above example, the contents of <y> are explicitly converted to the specified type before being assigned to <x>. Note: the actual contents of <y> remain unchanged.

Address assignment

```
x = &y;
```

This statement assigns the address of <y> to <x>.

Assembler pseudo-code:

```
LA   1,y
ST   1,x
```

Indirection

An asterisk is used to indicate indirection; namely, the associated variable contains an address. For example:

```
x = *y;
```

assigns the contents of the location pointed to by <y> to <x>.

Assembler pseudo-code:

```
L    1,y
MVC  x,Ø(1)
```

Further indirection

The number of levels of indirection is unlimited. An asterisk is used to indicate each level of indirection. For example:

```
int n, **ppi;

n = **ppi;
```

Assembler pseudo-code:

```
     L    1,ppi
     L    1,Ø(1)
     MVC  n,Ø(1)

ppi  DS   A
n    DS   F
```

Traversing chains

The number of levels of indirection is not normally in advance when chains are traversed. In practice, each chain element normally points to the next element with a zero address, indicating the end of the chain. In a simple case, a fixed number of indirection levels point to the associated data item. For example:

```
int i, *pi, **ppi;

pi = **ppi;
i = *pi;
```

Equivalent Assembler code (register 1 serves as <pi>):

```
      L    1,PPI
      MVC  I,Ø(1)

PPI   DS   A
I     DS   F
```

Building a chain

Building a chain involves the reverse process; the address of each element is assigned to the next lower pointer. For example:

```
int i, *pi, **ppi;


pi = &i;
ppi = &pi;
```

Equivalent Assembler code:

```
        LA    1,I
        ST    1,PI
        LA    1,PI
        ST    1,PPI

PI      DS    A
PPI     DS    A
I       DS    F
```

Static address assignments

The previous code fragments used dynamic address assignments where instructions are executed at execution time to assign the associated address. In many cases the address can be determined at compile-time (static address assignment), in which case the execution overhead associated with dynamic address assignment can be avoided.

Example:

```
int x, *px = &x;
```

This code fragment defines two fields, x and px, an integer and a pointer to an integer, respectively. This is obviously more efficient than the run-time equivalent (although a good compiler optimizer may be able to reduce this to static code):

```
int x, *px;

px = &x;
```

Note the difference between the two assignments: *px = &x and px = &x. In the first case, px is first defined as being a pointer to an integer (consequently, the asterisk) and then the field (namely px) is given the address of x.

AGGREGATES

Aggregates are data structures that consist of more than one elementary data item.

Struct (structure)

A C-struct can have two forms: declaration or definition (or a combination of both). See Figure 3.

Whereas a struct ‘declaration’ specifies the fields contained in the structure, and so approximates a DSECT, a struct ‘definition’ defines storage. Depending on whether the specified field is a pointer, either a storage area with the size of the structure or just a pointer (4 bytes in the 24/31-bit address world), which, depending on the instruction, may or may not contain a correct address.

Example 1, declaration of structure S:

```
struct S {
  int x;
```

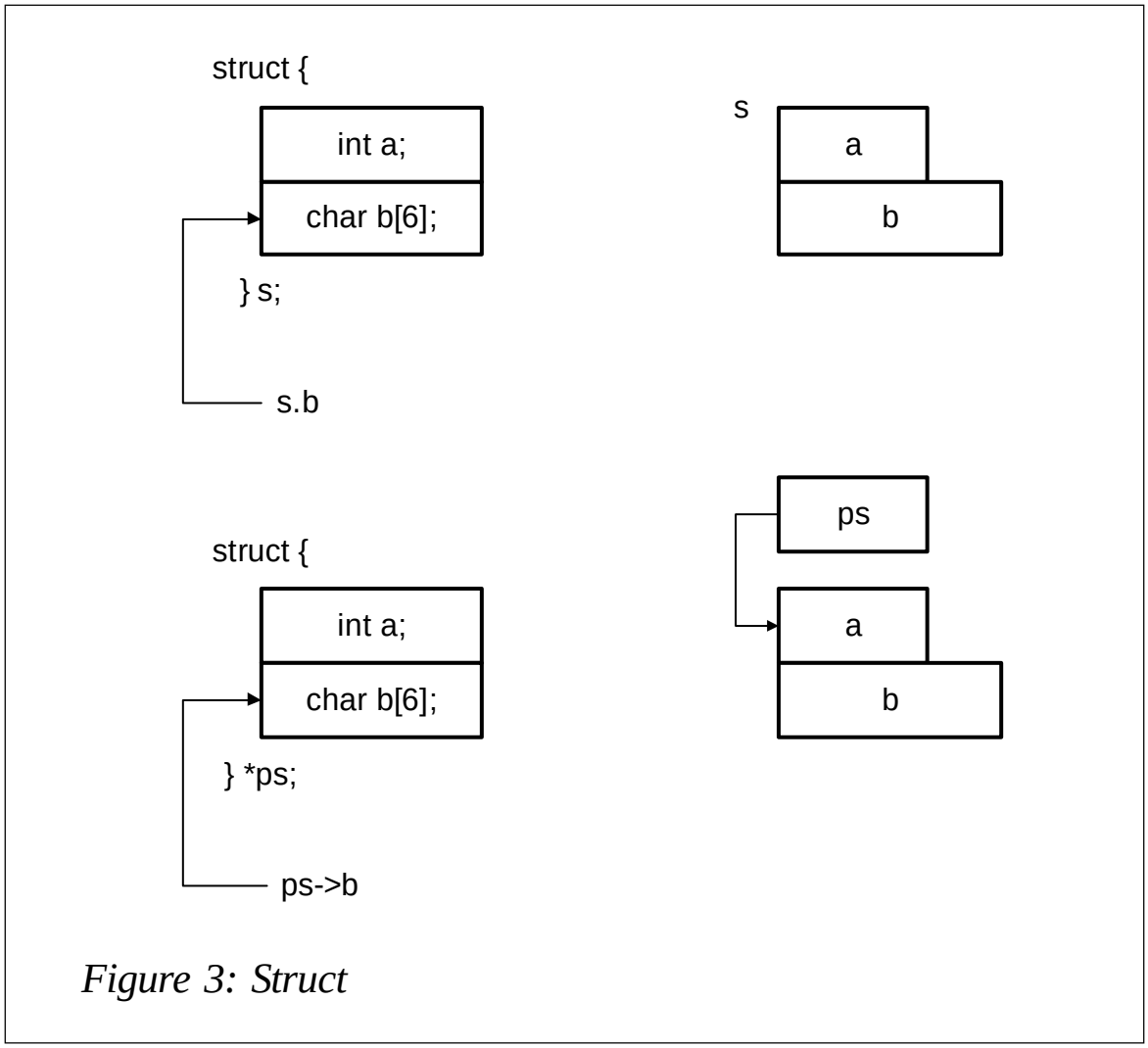


Figure 3: Struct

```
  char y[6];
};
```

Assembler pseudo-code:

```
S    DSECT
x    DS   F
y    DS   CL6
```

Example 2, definition of storage s:

```
struct {
  int x;
  char y[6];
} s;
```

Assembler pseudo-code:

```
s    DS   ØCL1Ø
x    DS   F
y    DS   CL6
```

Example 3, definition of a pointer to a structure ps:

```
struct {
  int x;
  char y[6];
} *ps;
```

Note: ps does not have any defined content here.

Assembler pseudo-code:

```
ps   DS   A(S)

S    DSECT
x    DS   F
y    DS   CL6
```

Example 4, definition of a pointer to a defined structure:

```
struct {
  int x;
  char y[6];
} s, *ps=&s;
```

Note: in this case, ps is assigned the address of s.

Assembler pseudo-code:

```
ps   DC   A(s)
```

```
s     DS   ØCL1Ø
x     DS   F
y     DS   CL6
```

Example 5, declaration of a structure with the associated pointer assigned at run-time:

```
struct S {
  int x;
  char y[6];
} s1, *ps;

ps = &s1;
```

This is the classic USING in Assembler:

```
      LA    1,S1
      USING S,1

S1    DS   CL1Ø
ps    DS   A

S     DSECT
x     DS   F
y     DS   CL6
```

Example 6, declaration of structure S used to define storage s1:

```
struct S {
  int x;
  char y[6];
} s1;
```

This corresponds to a dependent USING in Assembler:

```
      USING S1,S
S1    DS   XL1Ø

S     DSECT
x     DS   F
y     DS   CL6
```

Union (redefinition)

A C-union is a special form of a struct in which all the included fields occupy the same physical storage area. The longest field in the union defines the size of the storage area occupied by the union. As with structs, unions can have two forms: declaration or definition (or a combination of both). See Figure 4.

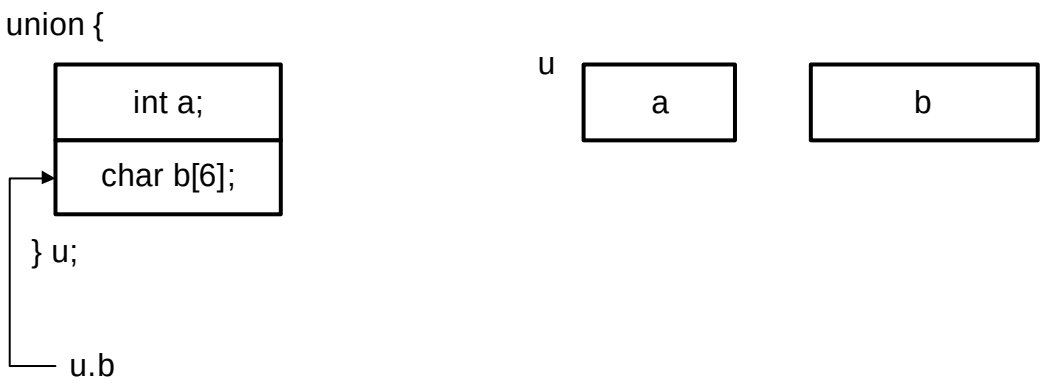


Figure 4: Union

Example 1, declaration of union U:

```
union U {
  int x;
  char y[6];
};
```

Assembler pseudo-code:

```
U    DSECT
x    DS   F
     ORG  U
y    DS   CL6
```

Example 2, definition of union u:

```
union {
  int x;
  char y[6];
} u;
```

Assembler pseudo-code:

```
u    DS   ØCL6
x    DS   F
     ORG  u
y    DS   CL6
```

Example 3, definition of a pointer to a union pu:

```
union U {
  int x;
  char y[6];
} *pu;
```

Assembler pseudo-code:

```
pu    DS    A(U)

U     DSECT
x     DS    F
      ORG   U    reset location counter back to U
y     DS    CL6
```

Example 4, definition of a pointer to a defined union:

```
union {
  int a;
  char y[6];
} u, *pu=&u;
```

Assembler pseudo-code:

```
pu    DC    A(u)

u     DS    ØCL6
a     DS    F
      ORG   u
y     DS    CL6
```

Accessing fields in a struct (or union)

To access the fields in the struct, they are specified as a qualified name: aggregate.name (the fullstop/period here is the structure-member operator). Depending on whether the aggregate is an actual definition or a declaration that is pointed to, this qualified name has the form aggregate.name or (*aggregate).name, respectively. The parentheses are required because the period delimiter has a higher priority than the asterisk delimiter. The structure-pointer operator (->) can be used instead of the indirection structure-member operator, namely, aggregateptr->name.

In the above description, aggregate and name refer to the aggregate (struct or union) and element within the associated aggregate, respectively.

Example 1:

```
struct {
  int x;
  char y[6];
```

```
} s, *ps = &s;

i = s.x;
i = ps->x;
i = (*ps).x;
```

In this case, the three assignments have an identical effect, namely the contents of element x of structure s are assigned to field i.

Example 2, declaration of structure S used to define two storage areas, s1 and s2, and transfer fields between these two storage areas.

```
struct S {
  int x;
  char y[6];
} s1, s2;

s1.x = s2.x;
```

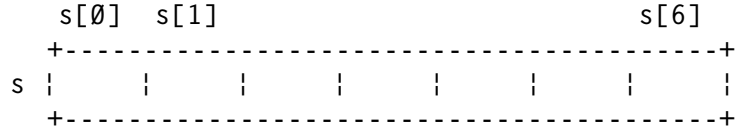
This corresponds to a labelled dependent USING in Assembler:

```
S1   USING SX,SX1
S2   USING SX,SX2
     MVC   S1.X,S2.X

SX1  DS   XL1Ø
SX2  DS   XL1Ø

SX   DSECT
x    DS   F
y    DS   CL6
```

STRINGS

A C-string is a special form of aggregate, namely an array of single characters, the elements of which are numbered starting at 0.

The string char s[7]; would have the following form:



Whereas in Assembler a string is a series of bytes with implicit

length, a string in C has a hexadecimal 0 (X'00') as (string) delimiter. The C run-time library provides a large number of string processing functions, which, however, use this string delimiter to determine the length of the fields being processed. Incorrect processing will result if it is not present – possibly data areas will be unintentionally overwritten (the infamous buffer overflow problem). This means external 'string' data (such as data from a file) may need to be moved to an internal buffer and appended with a string delimiter before it can be processed by string functions. The mem-family of functions (eg memcpy) or the bounded-string functions (eg strncpy) can be used to process a series of characters that do not have a string delimiter. However, in this case, the associated length must be specified explicitly.

Warning: whereas in Assembler the size of the receiving field defines the implicit length of a move (MVC) operation and so prevents unintentional overwriting, the length of the sending field determines the length of the move for string-copy functions in C.

C explicitly distinguishes between characters and strings, even though they both have the same elementary data type, namely char. The apostrophe (single quote) is used to specify a character, eg char c = 'a'. Quotation marks (paired quotes) are used to specify a string, eg char str[] = "abc". Note: in this case, str is actually 4 bytes, namely X'81828300' (in Assembler notation). As the use of square brackets implies, a string is actually an array of single characters, the first of which has address 0. Thus, the 4 bytes in str can be addressed with str[0], str[1], str[2], and str[3].

Example of processing an external 'string' (namely a series of characters without a string delimiter) using string functions:

```
char jobname[8];       /* external string */
char str[9];           /* buffer */
memcpy(str,jobname,8); /* move 8 characters */
str[8] = 0x00;         /* append string delimiter */
```

Note: for simplicity, this code fragment explicitly specifies the length as 8. As is also the case with Assembler, for maintenance reasons, explicit lengths should not be used. Whereas the sizeof() compile-time function can be used in place of the defined

length of the associated field, the strlen() run-time function determines the current length of the associated string (excluding the string delimiter).

For example, it would be better to write:

```
memcpy(str,jobname,sizeof(jobname));
```

Even better would be to make the field lengths dependent on each other (str must be (at least) one byte longer than jobname). For example:

```
#define LNAME 8
char jobname[LNAME];        /* external string (8 bytes) */
char str[LNAME+1];          /* buffer (9 bytes) */
memcpy(str,jobname,LNAME); /* move <jobname> */
str[LNAME] = 0x00;          /* append string delimiter */
```

Implicit address of string

When a string is specified in a function, not the string itself but, rather, the address of the string is implicitly used. Because a string is an array of single characters, the explicit address of element 0 (&str[0]) can be used as the address of the associated string (str).

For example:

```
char char3[] = "123";

strcpy(str,char3);
```

could also be written as

```
strcpy(&str[0],char3);
```

This has consequences when, for example, a struct is used in a function call:

```
struct CHAR3 {
  char byte1;
  char byte2;
  char byte3;
  char byte4;
} a = { '1', '2', '3', '\0'}; /* initialize */
```

This struct CHAR3 structure is in effect a string that has the same

contents as char3 in the previous fragment. However, because the address is not implicitly used for a struct in the strcpy() function call, the address-of operator (&) must be used explicitly (the cast (char *) is required to avoid a compiler warning):

```
strcpy(str,(char *)&a);
```

This is the case even when a struct contains a string. For example:

```
struct {
  char filler[4];
} a;

strcpy(a.filler,"abc"); /* initialize <filler> */
strcpy(str,(char *)&a);
```

Substring

To address a substring, set the appropriate element number.

For example, &str[2] addresses the character c of the string defined by char str[] = "abcd";

Example:

```
char str1[] = "123456";
char str2[] = "3456x";

int n;

n = memcmp(&str1[3],&str2[1],2); /* Ø = TRUE */
```

equivalent Assembler code:

```
str1   DC   C'123456'
str2   DC   C'3456x'

       CLC  str1+3(2),str2+1
```

or more generally using variables for the indexes and length:

```
int i, j, k, n;

n = memcmp(&str1[i],&str2[j],k);
```

equivalent Assembler code:

```
       L    2,I        index of <substr1>
```

```
        LA    2,str1(2)  address of <substr1>
        L     3,K        length of <substr1>
        L     4,J        index of <substr2>
        LA    4,str2(4)  address of <substr2>
        LR    5,3        length of <substr1, substr2>
        CLCL  2,4        compare <substr1> with <substr2>

I       DS    F
J       DS    F
K       DS    F
str1    DC    C'123456'
str2    DC    C'3456x'
```

EQUATES

To ease maintenance, programs should use direct values (numeric literals) only for invariant values, such as the number of days in a week (which is, and always will be, 7), incrementing a loop counter, but not the size of a table entry, etc. In all other situations it is better in Assembler to use an EQU (equate), for example, NTAB EQU 20 (where NTAB is the number of entries in a table). However, where possible, the size of a table entry, the size of the table, etc, should be determined by the Assembler based on the actual items, for example:

```
TABENTR  DSECT
PTR      DS    AL4
NAME     DS    CL8
LTAB     EQU  *-TABENTR      length of table entry (=12 bytes)

NTAB     EQU  2Ø             no. of table entries
TAB      DS   CL(NTAB*LTAB)  reserve space for table (24Ø bytes)
```

Similar techniques are also available in C. The #define preprocessor instruction can be used to define a symbol having the specified value, for example:

```
#define NTAB 2Ø
```

corresponds to:

```
NTAB  EQU  2Ø
```

in Assembler.

The following equivalent C instructions define the tab table with

NTAB entries with TABENTR structure:

```
#define NTAB 2Ø

struct TABENTR {
int ptr;
char name[8];
};

struct TABENTR tab[NTAB];
```

ADDRESS ARITHMETIC

In C, when an address pointer is incremented, it is incremented by the length of an associated data type. A union or a cast can be used if a specific length is required.

For example:

```
struct TABENTR {
  int ptr;
  char name[8];
} *pt;

union {
  struct TABENTR *pt;
  char *pc;
} u;

pt++; /* add one unit (=12 bytes) */

/* add one byte */
u.pt = pt;
u.pc += 1; /* force character-based arithmetic */
pt = u.pt;
```

DATA FIELD ALIGNMENT

In both Assembler and C, data fields are implicitly aligned to the associated boundary (characters to a byte boundary, fullword (int) to a word boundary, etc). The Assembler NOALIGN compile option forces byte-boundary alignment. In C, the _Packed qualifier can be used to force byte-boundary alignment for structure declarations.

Example:

```
struct S1 {
  char y;
  short dd;
} s1;

sizeof(s1) /* 4 */

_Packed struct S2 {
  char y;
  short dd;
} s2;

sizeof(s2) /* 3 */
```

Whereas s1 is 4 bytes long (a one-byte filler is inserted before dd to align it on a halfword boundary), s2 is only 3 bytes long.

BIT OPERATIONS

Bit testing

Many Assembler programs make wide use of the TM (Test under Mask) instruction, or the newer variants TMH (Test under Mask High) and TML (Test under Mask Low), to determine whether one or more bits (as specified in the mask) are set in the byte (halfword) being tested. The same effect can be achieved in C by performing an appropriate bit-And on the test byte and then testing the result for zero (bits not set) or non-zero (bits set) as appropriate.

The following Assembler code fragment:

```
         TM   B,X'80'
         JZ   NOTSET   bit not set

B        DC   X'81'
```

could be written in C as:

```
unsigned char b = 0x81;

n = b & 0x80;
if (n == 0) … /* bit not set */
```

These fragments test whether bit 0 (the high-order bit) of <b> is set (in which case <n> is non-zero).

Shifting

The C language provides operations that can be used to shift a field left or right.

The following Assembler code fragment:

```
         LHI   1,X'0120'
         SRL   1,5
```

could be written in C as:

```
unsigned short x = 0x0120;

x = x >> 5; /* 0x0009 */
```

CONVERSIONS

Typically for a high-level programming language, the C run-time library has some powerful functions that can be used for data conversion (formatting), although they are not as flexible as the equivalent Assembler machine instructions in some cases. Permitted numeric conversions are performed implicitly (for example, integer to float) or explicitly using a cast.

Editing

Editing involves converting a numeric value into a format that can be printed or displayed. The sprintf() function provides similar functionality as the powerful ED (Edit) Assembler instruction that converts a packed decimal number into display format using an edit mask.

Example:

```
int n = 123;
str char[6];

sprintf(str,"%05d ",n); /* 00123 */
```

The equivalent Assembler code fragment:

```
         MVC    STR,=X'F02120202020'
         ED     STR,N

N        DC     PL3'123'
STR      DS     CL6
```

Input conversion

In the reverse direction, character data is converted into numeric data. The code fragment shown assumes that the input data is valid and has the required format; the coding will fail if this is not the case. This section is included only to complement the previous section. The equivalent Assembler coding would be incomparably extensive.

For example, convert a fixed-point number (456.78) to integer. Because C does not support fixed-point arithmetic, the appropriate techniques must be used. The first method assumes that the input number has two decimal places. The second method is more flexible.

```
int n;
int ni, nd;
float nf;

char str[] = "456.78";
char strx[8];

sscanf(str,"%f",&nf); /* convert to float */
n = (int)(nf*100);    /* convert to fixed-point integer;
                         assume 2 decimal digits */

sscanf(str,"%d.%d",&ni,&nd); /* convert string to 2 integers */
n = sprintf(strx,"%d",nd);   /* no. of digits in decimal part */
nf = pow(10.0,n);            /* 10**n */
n = (ni*nf) + nd;
```

MACROS

Macros are used in Assembler to write reusable code that is resolved at compile time. IBM provides most interfaces to system services and control block definitions as Assembler macros. Similarly, most installations have a large macro library. The C preprocessor provides a similar, but less powerful,

functionality. C preprocessor instructions have # as the first character, for example #include, whereas Assembler macros are commonly used for both control block definitions and system service calls, C preprocessor instructions are principally used for field declarations (commonly known as header or include files).

Header files

The associated header files that contain the function prototypes, etc provided for the run-time library functions should always be specified, eg #include <math.h>, otherwise run-time errors can occur. Note: no compilation message is usually issued if the required header file is not specified.

CALLING PROGRAMS

Unlike COBOL, C and Assembler provide native support only for static program calls, although facilities exist for dynamic program loading (with the fetch() function and the LOAD LINK macro, respectively).

IBM MAINFRAME C COMPILER EXTENSIONS

The IBM mainframe C compiler has a number of extensions to handle special mainframe features. These major extensions are in the areas of input/output and decimal data fields. Because the details are too extensive, they are mentioned here only briefly.

Input/output extensions

The fopen() parameter list is extended in two ways. The first parameter, which normally specifies the file name, has been extended to allow a DD name to be specified, eg fopen("DD:OLDIN"),…). The second parameter has been extended to allow DCB parameters, etc, to be specified, eg fopen("DD:OLDIN","rb,type=record,recfm=*").

Decimal data type

The decimal data type allows fixed-point decimal arithmetic (with

implicit decimal point) to be performed. This simplifies the processing of packed decimal fields in files. The printf family of functions has been extended with the D data type to handle packed decimal fields.

For example:

```
decimal(7,2) dm;

dm = 47.11;
printf("dm:%D(7)\n",dm); /* 4711 */
printf("dm:%D(7,2)\n",dm); /* 47.11 */
```

PROGRAM EXAMPLE

The ubiquitous program that scans through the TIOT to list the jobname and the allocated DDnames is a good example that illustrates many of the techniques described above.

Assembler program

Note: judicious coding (in particular, Branch Relative (Jump) instructions) means this Assembler program does not require a base register. For simplicity, BAKR and PR are used for the program prologue and epilogue, respectively, although the usual program linkage could also be used.

```
GETTIOT  CSECT
GETTIOT  AMODE 31
GETTIOT  RMODE 24                RMODE(24) required for TPUT
         BAKR  14,Ø              save registers
         L     1,16              A(CVT): absolute address 16 (decimal)
         USING CVT,1
         L     9,CVTTCBP         A(TCB pointers)
         USING PTCB,9
         L     8,LASTTCB         A(current TCB)
         USING TCB,8
         L     7,TCBTIO          A(TIOT)
         USING TIOT1,7
         LA    1,TIOCNJOB        address of jobname
         TPUT  (1),L'TIOCNJOB    display jobname
         LA    2,TIOENTRY        first DDname entry
         USING TIOENTRY,2
LOOP     LA    1,TIOEDDNM        address of DDname
         CLI   Ø(1),C' '         blank entry
         JE    NEXT              ignore blank entry
```

```
         CLI   Ø(1),X'ØØ'          null entry
         JE    NEXT                ignore null entry
         TPUT  (1),L'TIOEDDNM      display DDname
NEXT     LA    1,Ø                 zeroise R1
         ICM   1,B'ØØØ1',TIOELNGH  length of current entry
         JZ    EOJ                 zero length = end of list
         AR    2,1                 update pointer
         J     LOOP                get next DD entry
EOJ      PR    ,                   program return
*
         CVT   DSECT=YES           map CVT
         IKJTCB ,                  map TCB
         IEFTIOT1 ,                map TIOT
*
PTCB     DSECT
NEXTTCB  DS    A                   address of next TCB to be dispatched
LASTTCB  DS    A                   address of last TCB to be dispatched
         END
```

The equivalent C program is shown below.

Note: to reduce the size of the declarations, the structures specify only those fields required in the application. Because the structure declarations are written as in-line code, the C program is larger than the Assembler program (which can use predefined macros for the control block definitions). In practice, the C structure declarations would normally be available as include (header) files.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef int PTR;

typedef struct {
  unsigned char tioelngh;     /* length of entry */
  unsigned char tioestta;     /* status byte A */
  unsigned char tioerloc[2];  /* relative location of pool */
           char tioeddnm[8];  /* DDname */
} TIOTENTRY;
TIOTENTRY *ptiotentry;

typedef struct {
  char tiocnjob[8];       /* job name */
  char tiocstep[8];       /* step name */
  char tiocjstn[8];       /* procedure job step name */
  TIOTENTRY tioentry;     /* DDname entry */
} TIOT;
```

```
TIOT *ptiot;

typedef struct {
  PTR tcbrbp;             /* address of the RB */
  PTR tcbpiea;            /* address of PIE */
  PTR tcbdeb;             /* address of queue DEBs */
  TIOT *tcbtio;           /* address of TIOT */
} TCB;
TCB *ptcb;

typedef struct {
  TCB *ptcbnext;          /* address of next TCB to be dispatched */
  TCB *ptcblast;          /* address of last TCB to be dispatched */
} PTCB;
PTCB *pptcb;

struct CVT {
  PTCB *cvttcbp;
} *pcvt, **ppcvt;

union {
  TIOTENTRY *ptiotentry;
  char *pc;
} temp;

#define LNAME 8           /* length of name field */
char name[LNAME+1];

int main()
{
  name[LNAME] = 0x00;       /* append X'00' to make C-string */

  /* start at CVT */
  ppcvt = (struct CVT**)16; /* absolute address of pointer to CVT;
                               cast to suppress warning */
  pcvt = *ppcvt;            /* pointer to CVT */
  pptcb = pcvt->cvttcbp;    /* pointer to TCB pointers */
  ptcb = pptcb->ptcblast;   /* pointer to current TCB */
  ptiot = ptcb->tcbtio;     /* pointer to TIOT */

  /* make C-string for jobname */
  memcpy(name,ptiot->tiocnjob,LNAME);
  printf("jobname: %s\n",name);

  /* scan through TIOT for DDnames */
  ptiotentry = &ptiot->tioentry;
  while (ptiotentry->tioelngh != 0) {
    /* make C-string for DDname */
    memcpy(name,ptiotentry->tioeddnm,LNAME);
    /* list only non-blank entry */
```

```
    if ((name[0] != ' ') & (name[0] != 0x00))
      printf("DDname: %s\n", name);
    /* use union to force char-based address arithmetic
       and suppress warning */
    temp.ptiotentry = ptiotentry;
    temp.pc += ptiotentry->tioelngh; /* address of next entry */
    ptiotentry = temp.ptiotentry;
  }
}
```

Passing arguments

C programs differ from Assembler programs not only in the form of arguments, but also the method used to pass arguments.

Both Assembler (and also COBOL and most other mainframe languages) and C have different conventional forms of the arguments depending on whether a main program or a subprogram (function) is called.

An Assembler main program receives the address (in register 1) of a pointer to the length-prefixed argument list. A C main program receives two parameters, the number of passed arguments and the array of pointers to the passed arguments (the calling argument is automatically parsed into blank-delimited subarguments).

For example: whereas in Assembler, arguments are normally passed by reference, in C, arguments are passed by value. This means, the address-of operator (&) must be used to pass an argument by reference (and exception are strings, which are always passed implicitly by reference).

```
int len;

void FUNC(int, int *);

int i;
int x1 = 1, x2 = 2;

int main(int argc, char *argv[]) {

for (i=0; i < argc; i++)
  printf("argv: %s\n", argv[i]);
```

```
FUNC(x1, &x2);
}

void FUNC(int i1, int *pi2) {
  printf("p1: %d\n", i1);
  printf("p2: %d\n", *pi2);
}
```

Anthony Rudd
Systems Programmer (Germany)

JES2 Health Monitor

INTRODUCTION

The JES2 Health Monitor is a new Z/OS 1.4 diagnostic tool, intended to be used when JES2 is non-responsive, to assist in determining why JES2 is not responding to requests.

A new separate address space, JES2MON, collects data and accepts commands to summarize JES2 problems. It does this through the use of highlighted messages and operator commands. Messages are issued when conditions exist that can seriously impact JES2 performance.

JES2MON ADDRESS SPACE

The JES2 Health Monitor runs in a separate address space from JES2. Its address space name is JES*x*MON, where JES*x* is the name of the subsystem that is being monitored.

The JES2 Health Monitor does not require any special set-up: there are no initialization statements.

The monitor cannot be turned off. The overhead associated with the address space is minimal (less than 1% of a CPU).

JES*x*MON is automatically started as part of the JES2 initialization

and stopped during clean JES2 shutdown. It is automatically restarted in the case of monitor failures.

Within the monitor address space, a number of tasks perform the functions needed to determine how the JES2 address space is performing:

- Main task – starts and stops the JES*x*MON address space.
- Sampler – samples the JES2 TCB resource address space.
- Probe – examines sampled data and issues alerts.
- Command – processes operator commands.

The monitor uses cross memory services to examine data in the JES2 address space. Data is examined both by sampling values at regular intervals and by extracting data needed to build command responses.

JES2 HEALTH MONITOR ALERTS MANAGEMENT

The JES2 Health Monitor examines the activity of the JES2 main task, looking for conditions that could indicate a problem.

If one condition is detected, the JES2 Health Monitor issues highlighted messages:

```
$HASP9201 JES2 MAIN TASK WAIT DETECTED AT module+offset
$HASP9202 POTENTIAL JES2 MAIN TASK LOOP DETECTED NEAR module+offset
$HASP9203 LONG PCE DISPATCH
$HASP9204 JES2 MAIN TASK BUSY
$HASP9205 PCE WAITING FOR BERT LOCK
$HASP9206 PCE WAITING FOR JOB LOCK
$HASP9207 JES2 CHECKPOINT LOCK HELD
$HASP9208 JES2 MAIN TASK LOCAL LOCK WAIT AT module+offset
$HASP9209 JES2 MAIN TASK NON-DISPATCHABLE AT module+offset
$HASP9210 JES2 MAIN TASK PAGING WAIT AT module+offset
$HASP9211 JES2 MAIN TASK NOT RUNNING
```

These conditions can occur normally when JES2 is running. However, if the condition persists for a long enough period of time, it may indicate a real problem.

Once a condition is detected that needs monitoring, it is considered an incident.

Incidents are divided into three categories, depending on the duration of the incident.

- Normal processing – below a low time threshold (less than 5 seconds), incidents are considered part of normal processing.
- Tracks – once that low time interval is exceeded, the incident is tracked.
- Alerts – these are issued when an incident that is being tracked crosses a second threshold. This threshold is not a time threshold but rather a sampling threshold. Before an incident becomes an alert, a specific number of samples must be collected indicating that the condition still exists.

Once an incident is considered an alert, a highlighted message is issued to the console. If the alert persists, the highlighted message is re-issued at regular intervals. When the condition that caused the alert no longer exists, the highlighted message is deleted. Sample alerts:

```
17.38.14 *$HASP920Ø POTENTIAL JES2 MAIN TASK LOOP DETECTED NEAR
HASTDIAG+Ø1FC4E
DURATION-ØØØ:ØØ:15.77 PCE-COMM EXIT-NONE JOB ID-JOBØØØ11
COMMAND-$TJ1-*,LOOP1=1
17.38.14 *$HASP920Ø7 JES2 CHECKPOINT LOCK HELD                          -
DURATION-ØØØ:ØØ:15.87
17.38.46 *$HASP920Ø2 POTENTIAL JES2 MAIN TASK LOOP DETECTED NEAR
HASTDIAG+Ø1FC4E
DURATION-ØØØ:ØØ:47.62 PCE-COMM EXIT-NONE JOB ID-JOBØØØ35
COMMAND-$TJ1-*,LOOP1=1
17.38.46 *$HASP920Ø7 JES2 CHECKPOINT LOCK HELD                         -
DURATION-ØØØ:ØØ:47.68
```

JES2 HEALTH MONITOR NOTICES MANAGEMENT

In addition to alerts and tracks, the monitor can display information on conditions that are not time related in nature. These are called notices, and they are displayed only in response to $JD STATUS and $JD JES commands. Notices are conditions that arise from operator commands, resource shortages, or system errors.

Possible notice messages generated by the $JD JES command are:

```
$HASP915Ø {NO} JES2 NOTICES
$HASP9151 JES2 ADDRESS SPACE NOT ACTIVE
$HASP9152 JES2 INITIALIZING
$HASP9153 JES2 TERMINATING
$HASP9154 CKPT RECONFIGURATION IN PROGRESS
$HASP9155 ADDRESS SPACES WAITING FOR INTERNAL READERS
$HASP9156 ADDRESS SPACES WAITING FOR SPOOL SPACE
$HASP9157 CANNOT RESTART JES2, IPL REQUIRED
$HASP9158 JES2 PROCESSING STOPPED, $S NEEDED
$HASP9159 JES2 EXECUTION PROCESSING STOPPED ($PXEQ)
$HASP916Ø AT LEAST ONE PCE HAS ENDED
$HASP9161 NOT ALL SPOOL VOLUMES ARE AVAILABLE
$HASP9162 PCES WAITING FOR SPOOL SPACE
$HASP9163 FAST SPOOL GARBAGE COLLECTION (SPOOLDEF GCRATE=FAST)
```

JES2 HEALTH MONITOR COMMANDS

The JES2 Health Monitor supports a number of commands to display information it is tracking.

All monitor commands start with the JES2 command prefix followed by a ‘J’. Messages generated by the JES2 Health Monitor are $HASP9*xxxx*.

$JD STATUS

The $JD STATUS command is the primary diagnostic command for the monitor.

When a problem is suspected in JES2, the $JD STATUS command should be used to determine whether the monitor has detected anything.

It displays the current status of the JES2 being monitored. This display includes all alerts and notices that are current:

```
$JD STATUS
$HASP912Ø D STATUS 613
$HASP9121 NO OUTSTANDING ALERTS
$HASP915Ø JES2 NOTICES
$HASP9159 JES2 EXECUTION PROCESSING STOPPED ($PXEQ)
```

$JD JES

The $JD JES command displays events that are being tracked by the JES2 Health Monitor but are not necessarily problems:

```
$JD JES
$HASP912Ø D JES 837
$HASP9121 NO OUTSTANDING ALERTS
$HASP9122 NO INCIDENTS BEING TRACKED
$HASP915Ø JES2 NOTICES
$HASP9151 JES2 ADDRESS SPACE NOT ACTIVE
```

$JD DETAILS

The $JD DETAILS command displays current statistics for resource utilization and CPU sample statistics. The MVS wait table is also displayed:

```
$JD DETAILS
$HASP91Ø3 D DETAIL Ø67
$HASP91Ø4 JES2 RESOURCE USAGE SINCE 2Ø04.Ø89 11:11:52
RESOURCE    LIMIT    USAGE      LOW     HIGH  AVERAGE
-------- -------- -------- -------- -------- --------
BERT         21ØØ     1931     1931     2Ø23     2Ø15
BSCB           1Ø        Ø        Ø        Ø        Ø
BUFX          3ØØ        8        Ø       11        2
CKVR           17        Ø        Ø        Ø        Ø
CMBS          113        Ø        Ø       82        Ø
CMDS          1ØØ        Ø        Ø        Ø        Ø
ICES           43        Ø        Ø        Ø        Ø
JNUM         9999      958      958     1ØØØ      999
JOES         5ØØØ       22       22       52       5Ø
JQES         1ØØØ      958      958     1ØØØ      999
LBUF          3ØØ        Ø        Ø        Ø        Ø
NHBS          123        Ø        Ø        Ø        Ø
SMFB            8        Ø        Ø        2        Ø
TGS          25ØØ     1Ø25     1Ø25     1Ø67     1Ø59
TTAB            3        Ø        Ø        Ø        Ø
VTMB           38        Ø        Ø        Ø        Ø
$HASP91Ø5 JES2 SAMPLING STATISTICS SINCE 2Ø04.Ø89 11:11:52
TYPE               COUNT  PERCENT
---------------  ------  ------
ACTIVE               613     3.59
IDLE               16173    94.92
LOCAL LOCK             Ø     Ø.ØØ
NON-DISPATCHABLE       Ø     Ø.ØØ
PAGING                 Ø     Ø.ØØ
OTHER WAITS          251     1.47
TOTAL SAMPLES      17Ø37
```

```
$HASP9106 JES2 MAIN TASK MVS WAIT TABLE
DATE     TIME     ADDRESS  MODULE   OFFSET WT-COUNT SM-COUNT PCE XIT
-------- -------- -------- -------- ------ -------- -------- -- --
2004.089 11:19:58 013B6D6C ISGGWAIT+00014C      132      132  13 JCO
2004.089 11:11:59 00FECDEE IEAVEWAT+0007DE        6       15 MLT JCO
2004.089 11:06:37 00032558 HASPTERM+005538        1      127  10 JCO
2004.089 11:06:44 0002F1DC HASPTERM+0021BC        3       12  10 JCO
2004.089 11:11:52 049CF748 IGC0013I+00C748        1        1  23 JCO
2004.089 11:11:53 0004235E HASPCKPT+0072D6        1        4  23 JCO
2004.089 11:11:55 09207DA2 HASPIRDA+002892        1        3  23 JCO
2004.089 11:11:55 00140774 HASPCKRR+00211C        1        2  23 JCO
2004.089 11:11:56 015198FE IEWFETCH+00152E        5        8  23 JCO
2004.089 11:11:57 00144FA2 HASPDYN +000F7A        1        7  23 JCO
```

$JD MONITOR

The $JD MONITOR command displays status information for each of the monitor subtasks and service information for each module that makes up the monitor:

```
$HASP9100 D MONITOR 311
NAME     STATUS       ALERTS
-------- ------------ ------------------------
MAINTASK ACTIVE
SAMPLER  ACTIVE
COMMANDS ACTIVE
PROBE    ACTIVE
$HASP9102 MONITOR MODULE INFORMATION
NAME     ADDRESS  LENGTH   ASSEMBLY DATE  LASTAPAR LASTPTF
-------- -------- -------- -------------- -------- --------
HASJMON  0F129000 00001088 04/15/02 10.33 NONE     NONE
HASJSPLR 0F12D1C8 00002838 04/15/02 10.33 NONE     NONE
HASJCMDS 0F12A088 00003140 11/08/02 07.50 OW56481  UW95451
```

$JD HISTORY

The $JD HISTORY command displays up to 72 hours of resource utilization and CPU sample statistics:

```
$JD HISTORY

$HASP9131 JES2 TGS  USAGE HISTORY
DATE     TIME        LIMIT    USAGE      LOW     HIGH  AVERAGE
-------- -------- -------- -------- -------- -------- --------
2004.089 14:00:00     2500      349      347      349      347
2004.089 13:00:00     2500      347      347      348      347
2004.089 12:00:00     2500      348      348      348      348
2004.089 11:11:52     2500      348      346     1067      490
2004.089 10:44:51     2500     1067     1066     1067     1067
```

$J STOP

This command shuts down the JES*x*MON address space.

If the JES2 address space is active, the monitor address space is restarted automatically. This command will clear any history the monitor has been maintaining.

```
$J STOP
$HASP91Ø1 MONITOR STOPPING 571
$HASP9Ø85 JES2 MONITOR ADDRESS SPACE STOPPED FOR JES2
IEF4Ø4I IEESYSAS - ENDED - TIME=1Ø.44.49
IEF196I IEF142I IEESYSAS JES2MON - STEP WAS EXECUTED - COND CODE ØØØØ
IEF196I IEF373I STEP/IEFPROC /START 2ØØ4Ø86.1238
IEF196I IEF374I STEP/IEFPROC /STOP  2ØØ4Ø89.1Ø44 CPU    4MIN 37.79SEC
IEF196I SRB    3MIN 26.44SEC VIRT     4K SYS   14ØK EXT   68224K SYS
IEF196I 1Ø632K
IEF196I IEF375I  JOB/JES2MON /START 2ØØ4Ø86.1238
IEF196I IEF376I  JOB/JES2MON /STOP  2ØØ4Ø89.1Ø44 CPU    4MIN 37.79SEC
IEF196I SRB    3MIN 26.44SEC
$HASP9Ø84 JES2 MONITOR ADDRESS SPACE STARTED FOR JES2
```

Alexandre Goupil
Systems Programmer (France)

October 2002 – September 2004 index

Items below are references to articles that have appeared in *MVS Update* since Issue 193, October 2002. References show the issue number followed by the page number(s). Subscribers can download copies of all issues in Acrobat PDF format from Xephon's Web site.

Cybermation has announced Version 5.4 of ESP Workload Manager, its job scheduling software. The product has workload management tools that help enterprises achieve greater efficiency and optimize job scheduling in complex environments. This release includes real-time workload balancing across the enterprise to maximize server utilization and improve throughput.

With its new enterprise-wide workload balancing feature, ESP Workload Manager makes real-time decisions on the distribution of workload based on server capacity. Once a decision is made it directs workload via Cybermation's ESP System Agent technology to the server with the highest available capacity to handle it.

This release includes procedure cacheing for data centres with large applications, which allows the software to work faster and reduce I/O. Other enhancements include multi-threading of the ESP subsystem, additional display fields for viewing overdue jobs, a new mailbox service for enhanced message delivery option, support for six-digit JES numbers, and support for distributed job names exceeding eight characters.

For further information contact:
Cybermation, 125 Commerce Valley Drive West, 8th Floor, Markham, ON L3T 7W4, Canada.
Tel: (905) 707 4400.
URL: http://www.cybermation.com/products/jobscheduling/overview.html.

* * *

TeamQuest has announced that it is providing support for the IBM zSeries with a comprehensive SuSE Linux solution in TeamQuest Performance 9.1.

New features released in Version 9.1 of TeamQuest Performance include additional agents and multi-system modelling, with the ability to consolidate workloads across servers. Automated linear trend and short-trend analysis capabilities help identify over-utilized servers and detect drifts in variables at an early stage and predict future behaviour based on statistical techniques. A new interface allows users to manage and administer all systems from a single console, enabling users to define policies to span several systems.

For further information contact:
TeamQuest, One TeamQuest Way, Clear Lake, IA 50428, USA.
Tel: (641) 357 2700.
URL: http://www.teamquest.com/pressroom/pr/0604-2.shtml.

* * *

Nexsan Technologies and Luminex Software have announced a storage platform that allows mainframes to become servers on a Fibre Channel SAN through the combination of Luminex's Virtual|BLUE 3990 DASD Control Unit or Virtual|BLUE VTS Tape Control Unit and Nexsan's disk storage arrays.

Virtual|BLUE 3990 and VTS allow IBM mainframes to connect directly to Nexsan's open system RAID arrays via native ESCON I/O channels.

The company claims that the Nexsan/Luminex platform is one-fifth the cost of similar capacity traditional mainframe storage systems.

For further information contact:
Nexsan Technologies, 21700 Oxnard St, Suite 1850, Woodland Hills, CA 91367, USA.
Tel: (818) 715 9111.
URL: http://www.nexsan.com.