198

MVS update

March 2003

In this issue

Published by

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

North American office

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

Subscriptions and back-issues

A year's subscription to *MVS Update*, comprising twelve monthly issues, costs £340.00 in the UK; $505.00 in the USA and Canada; £346.00 in Europe; £352.00 in Australasia and Japan; and £350.00 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 1999 issue, are available separately to subscribers for £29.00 ($43.50) each including postage.

MVS Update on-line

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

Editor

Trevor Eddolls

Disclaimer

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

Contributions

When Xephon is given copyright, articles published in *MVS Update* are paid for at the rate of £170 ($260) per 1000 words and £100 ($160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 ($80) per 100 lines. In addition, there is a flat fee of £30 ($50) per article. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

Printed in England.

Resolving PDSE problems

Sometimes the PDSE (partitioned dataset extended) subsystem causes problems with PDSE datasets. What you see is everything frozen – jobs and users working with datasets have a long wait. But you should know that PDSE is the problem.

With the V SMS,PDSE,ANALYSIS command, you can determine whether PDSE does have problems.

ANALYSING THE STATE OF THE PDSE SUBSYSTEM

Use the VARY SMS,PDSE,ANALYSIS command to determine the state of the PDSE subsystem. You can run the analysis on all the PDSEs that are open, or you can specify a particular PDSE by dataset name and, optionally, the volser.

Consult *z/OS DFSMSdfp Diagnosis Reference* for specific information about how to use this command, including the command syntax.

If the above command shows 'latches', you can pick up latchaddr, asid, and tcbaddr information from the console/system log and use:

```
V SMS,PDSE,FREELATCH(latchaddr,asid,tcbaddr)
```

RELEASING PDSE LATCHES

Use the VARY SMS,PDSE,FREELATCH command to release a latch that the V SMS,PDSE,ANALYSIS command has determined is frozen.

Alternatively, you can use UNLATCH, a REXX EXEC that executes V SMS,PDSE,ANALYSIS, checks the output, and generates V SMS,PDSE,FREELATCH(latchaddr,asid,tcbaddr) with the appropriate values and prompts whether to execute the generated command.

TSO UNLATCH EXECUTION

UNLATCH and MVSCMD must be in a SYSEXEC or SYSPROC concatenated dataset.

UNLATCH

```
/* REXX - (c) Klaus Bouschen, K.I.S.S. Consulting GmbH              */
/* mail: KLAUS (at) BOUSCHEN.DE                                    */
/* UNLATCH: ANALYSIS and FREELATCH PDSE-Problems                   */
/* RACF: MVS.VARY.SMS                                 UPDATE       */
/* RACF: TSOAUTH/CONSOLE                              READ         */
x=outtrap("C.","*")
"MVSCMD VARY SMS,PDSE,ANALYSIS"
x=outtrap("OFF")
do i=1 to c.Ø
 select
  when pos("++ no exceptional dataset conditions detected",c.i)¬=Ø
   then
    do
     say ""
     say "++ no exceptional dataset conditions detected"
     exit
    end
  when pos("++ no PDSEs connected",c.i)¬=Ø
   then
    do
     say ""
     say "++ no PDSEs connected"
     exit
    end
  when pos("++ Unable to latch DIB Hash Table Latch:",c.i)¬=Ø
   then
    do
     parse var c.i . "++ Unable to latch DIB Hash Table Latch:"latch,
          "Holder("asid":"tcb")" .
     latch=strip(latch);asid=strip(asid);tcb=strip(tcb)
     say "+-------------------------------------------------+"
     say "!" left("LATCH:"latch "ASID:"asid "TCB:"tcb,5Ø)     "+"
     say "+-------------------------------------------------+"
     say "! Execute FREELATCH-Command?                       +"
     say "!                                Y => Yes, execute   +"
     say "!                         otherwise => No,  exit     +"
     say "+-------------------------------------------------+"
     pull answer
     if answer="Y" then
      "MVSCMD VARY SMS,PDSE,FREELATCH("latch","asid","tcb")"
     exit
    end
```

```
  otherwise
   do
    if pos("++",c.i)¬=Ø then
     say "Output of VARY SMS,PDSE,ANALYSIS could not be interpreted"
    else
     say c.i
   end
 end
end
```

MVSCMD

MVSCMD executes and traps system commands, called by UNLATCH. (By the way, you can use it as a single command, ie TSO MVSCMD D A,L.)

```
/* REXX - (c) Klaus Bouschen, K.I.S.S. Consulting GmbH              */
/* mail: KLAUS (at) BOUSCHEN.DE                                    */
/* MVSCMD: Execute CONSOLE-Command (SOLDSIPLAY=NO, UNSOL=YES)      */
/*         TSOAUTH/CONSOLE  Read-Access required                   */
/*         TSOAUTH/CONSPROF Read-Access required                   */
arg command
if command="" then
 do
  say "+-------------------------+"
  say "! Type in CONSOLE-Command !"
  say "!                         !"
  say "!          no input->exit !"
  say "+-------------------------+"
  pull command
  if command="" then exit
 end
"CONSPROF SOLDISPLAY(NO) UNSOLDISPLAY(NO)",
        "SOLNUM(3ØØØ) UNSOLNUM(3ØØØ)"
"CONSOLE SYSCMD("COMMAND") CART('MVSCMD  ') NAME("userid()"C)"
run=1
get_message:
rc_getmsg=getmsg("C.","SOL","MVSCMD  ",,8) /* wait max. 8 secs */
if rc_getmsg¬=Ø then
 do
  if run=1 then say "GETMSG RC="rc_getmsg
             else nop
 end
else
 do
  do i=1 to c.Ø
   say '>>' c.i  /* here comes response to your command, if in time */
  end
```

```
  run=run+1
  signal get_message
 end
"CONSOLE DEACTIVATE"
```


A utility that uses the IMWRESET WLM API to interact with WLM

INTRODUCTION

The IWMRESET WLM macro allows the caller to perform the same functions as with the RESET system command.

When the system is running in WLM goal mode, the caller can:

- Change the service class of work currently in execution, with RESET.
- Quiesce work currently in execution, with the QUIESCE keyword.
- Resume quiesced work with the RESUME keyword.

The RESETWSO Assembler program uses the IWMRESET API to interact with WLM.

RESETWSO SOURCE CODE

```
RESETWSO CSECT
RESETWSO AMODE 31
RESETWSO RMODE ANY
*
* VALID SYNTAX:
* -------------
*
* PARM='RESET,ASNAME,SRVCLASS'
* PARM='QUIESCE,ASNAME'
```



```
* PARM='RESUME,ASNAME'
*
* REGISTER USAGE:
*
* R12 = BASE REGISTER
*
         SAVE  (14,12)
         BASR  R12,Ø
         USING *,R12               R12 = BASE REGISTER
         L     R2,Ø(R1)            LOAD PARAMETER ADDRESS
         LH    R3,Ø(R2)            LENGTH
         GETMAIN R,LV=WORKL
         ST    R1,8(R13)
         ST    R13,4(R1)
         LR    R13,R1
         USING WORK,R13
         LR    R6,R3               R6 = COMMAND LENGTH
*
*        LINK    EP=SHOWREGS
*
         MVI   XCMD,C' '           CLEAR RECEIVING AREA
         MVC   XCMD+1(L'XCMD),XCMD
         BCTR  R3,Ø                SUBTRACT 1 FROM LENGTH FOR MOVE
         EX    R3,MOVE             MOVE COMMAND FROM PARAMETER LIST
*
         MVC   WTOA(WTOL),WTOLIST
         MVC   WTOM,=CL8Ø''
         MVC   WTOM+ØØ(Ø8),PGMNAME
         MVC   WTOM+Ø8(12),=CL12' - COMMAND: '
         MVC   WTOM+2Ø(4Ø),XCMD
         L     R2,=A(WTOMLEN)
         STH   R2,WTOML
         LA    R2,WTOMSG
         WTO   TEXT=(R2),MF=(E,WTOLIST)
         MVC   FUNCTION,=CL8''
         MVC   ASNAME,=CL8''
         MVC   SRVCLASS,=CL8''
         SR    R8,R8
*
* PARSE REQUESTED FUNCTION
*
         LA    R4,XCMD
         LA    R5,FUNCTION
LF1      EQU   *
         CLC   Ø(1,R4),=CL1','
         BE    OKF1
         MVC   Ø(1,R5),Ø(R4)
         LA    R7,1
         AR    R5,R7
         AR    R4,R7
```

```
         AR    R8,R7
         CR    R8,R6               END OF COMMAND ?
         BH    KO1                 YES, INVALID COMMAND
         B     LF1
*
OKF1     EQU   *
*
* PARSE ADDRESS SPACE NAME
*
         LA    R7,1
         AR    R4,R7
         AR    R8,R7
         CR    R8,R6               END OF COMMAND ?
         BH    KO1                 YES, INVALID COMMAND
         LA    R5,ASNAME
*
LA1      EQU   *
*
         CLC   Ø(1,R4),=CL1','
         BE    OKA1
         MVC   Ø(1,R5),Ø(R4)
         LA    R7,1
         AR    R5,R7
         AR    R4,R7
         AR    R8,R7
         CR    R8,R6               END OF COMMAND ?
         BH    COEND               YES
         B     LA1
*
OKA1     EQU   *
*
         CLC   FUNCTION,=CL8'RESET'
         BNE   COEND
*
* PARSE SRVCLASS
*
         LA    R7,1
         AR    R4,R7
         AR    R8,R7
         CR    R8,R6               END OF COMMAND ?
         BH    KO1                 YES, INVALID COMMAND
         LA    R5,SRVCLASS
*
LS1      EQU   *
*
         CLC   Ø(1,R4),=CL1','
         BE    OKS1
         MVC   Ø(1,R5),Ø(R4)
         LA    R7,1
         AR    R5,R7
```

```
         AR    R4,R7
         AR    R8,R7
         CR    R8,R6               END OF COMMAND ?
         BH    COEND               YES
         B     LS1
OKS1     EQU   *
COEND    EQU   *
         MVC   WTOA(WTOL),WTOLIST
         MVC   WTOM,=CL8Ø''
         MVC   WTOM+ØØ(Ø8),PGMNAME
         MVC   WTOM+Ø8(13),=CL13' - FUNCTION: '
         MVC   WTOM+21(Ø8),FUNCTION
         MVC   WTOM+28(11),=CL11' - ASNAME: '
         MVC   WTOM+39(Ø8),ASNAME
         CLC   SRVCLASS,=CL8''
         BE    NOSRVC
         MVC   WTOM+47(13),=CL13' - SRVCLASS: '
         MVC   WTOM+6Ø(Ø8),SRVCLASS
NOSRVC   EQU   *
         L     R2,=A(WTOMLEN)
         STH   R2,WTOML
         LA    R2,WTOMSG
         WTO   TEXT=(R2),MF=(E,WTOLIST)
         CLC   FUNCTION,=CL8'RESET'
         BE    RESET
         CLC   FUNCTION,=CL8'QUIESCE'
         BE    QUIESCE
         CLC   FUNCTION,=CL8'RESUME'
         BE    RESUME
*
RESET    EQU   *
*
         AUTHON
         MODESET KEY=ZERO,MODE=SUP
         IWMRESET JOBNAME=ASNAME,                                    X
               USERID=USERID,                                        X
               PRODUCT=PRODUCT,                                      X
               FUNCTION=RESET,                                       X
               SRVCLASS=SRVCLASS,                                    X
               RETCODE=RETCODE,                                      X
               RSNCODE=RSNCODE
         LTR   R15,R15
         BNZ   ERROR
         MODESET KEY=NZERO,MODE=PROB
         AUTHOFF
         LA    R15,Ø
         ST    R15,RC                  SAVE RC
         B     RETURN
*
QUIESCE  EQU   *
```

```
*
         AUTHON
         MODESET KEY=ZERO,MODE=SUP
         IWMRESET JOBNAME=ASNAME,                                      X
               USERID=USERID,                                          X
               PRODUCT=PRODUCT,                                        X
               FUNCTION=QUIESCE,                                       X
               RETCODE=RETCODE,                                        X
               RSNCODE=RSNCODE
         LTR   R15,R15
         BNZ   ERROR
         MODESET KEY=NZERO,MODE=PROB
         AUTHOFF
         LA    R15,Ø
         ST    R15,RC                  SAVE RC
         B     RETURN
*
RESUME   EQU   *
*
         AUTHON
         MODESET KEY=ZERO,MODE=SUP
         IWMRESET JOBNAME=ASNAME,                                      X
               USERID=USERID,                                          X
               PRODUCT=PRODUCT,                                        X
               FUNCTION=RESUME,                                        X
               RETCODE=RETCODE,                                        X
               RSNCODE=RSNCODE
         LTR   R15,R15
         BNZ   ERROR
         MODESET KEY=NZERO,MODE=PROB
         AUTHOFF
         LA    R15,Ø
         ST    R15,RC                  SAVE RC
         B     RETURN
ERROR    EQU   *
         MODESET KEY=NZERO,MODE=PROB
         AUTHOFF
         MVC   CL2,RETCODE+2        LAST 2 DIGITS = RC
         MVC   CL3,=XL3'Ø'          2 + 1 = FOR SIGN
         MVC   CL3(2),CL2
         UNPK  CL6,CL3
         MVC   CL2,CL6+3            TO GET RC VALUE
         NC    CL2,=XL2'ØFØF'
         TR    CL2,T_HEX
         MVC   RETC,CL2             RC = XX
         MVC   CL2,RSNCODE+2        LAST 2 DIGITS = RSN
         MVC   CL3,=XL3'Ø'          2 + 1 = FOR SIGN
         MVC   CL3(2),CL2
         UNPK  CL6,CL3
         MVC   CL4,CL6+1            TO GET RSN VALUE
```

```
         NC    CL4,=XL4'ØFØFØFØF'
         TR    CL4,T_HEX
         MVC   RSNC,CL4              RSN = XXXX
         MVC   WTOA(WTOL),WTOLIST
         MVC   WTOM,=CL8Ø''
         MVC   WTOM+ØØ(Ø8),PGMNAME
         MVC   WTOM+Ø8(2Ø),=CL2Ø' - ERROR - IWMRESET'
         MVC   WTOM+28(Ø8),=CLØ8' RC = XX'
         MVC   WTOM+34(Ø2),RETC
         CLC   RETC,=CL2'Ø4'         RC = 4 ?
         BE    IMSG                  YES, NO RSNCODE
         MVC   WTOM+36(13),=CL13' / RSN = XXXX'
         MVC   WTOM+45(Ø4),RSNC
IMSG     EQU   *
         L     R2,=A(WTOMLEN)
         STH   R2,WTOML
         LA    R2,WTOMSG
         WTO   TEXT=(R2),MF=(E,WTOLIST)
         B     RC8
KO1      EQU   *
         MVC   WTOA(WTOL),WTOLIST
         MVC   WTOM,=CL8Ø''
         MVC   WTOM+ØØ(Ø8),PGMNAME
         MVC   WTOM+Ø8(2Ø),=CL2Ø' - INVALID COMMAND '
         L     R2,=A(WTOMLEN)
         STH   R2,WTOML
         LA    R2,WTOMSG
         WTO   TEXT=(R2),MF=(E,WTOLIST)
RC8      EQU   *
         LA    R15,8
         ST    R15,RC                  SAVE RC
RETURN   EQU   *
         L     R3,RC                   RESTORE RC
         L     R13,4(R13)              RESTORE R13
         L     R1,8(R13)
         FREEMAIN R,LV=WORKL,A=(R1)
         LR    R15,R3                  SET UP RC
         L     R14,12(R13)
         LM    RØ,R12,2Ø(R13)
         BSM   Ø,R14                   RETURN TO MVS AND USE RC=R15
MOVE     MVC   XCMD(Ø),2(R2)        MOVE PARM (OFFSET 2 / LENGTH)
PGMNAME  DC    CL8'RESETWSO'
USERID   DC    CL8'RESETWSO'
PRODUCT  DC    CL8'RESETWSO'
T_HEX    DC    X'FØF1F2F3F4F5F6F7F8F9C1C2C3C4C5C6'
WTOLIST  WTO   TEXT=,ROUTCDE=11,MF=L
WTOL     EQU   *-WTOLIST
WORK     DSECT
SAVEAREA DS    18F
WTOA     DS    CL(WTOL)
```

```
FUNCTION DS    CL8
ASNAME   DS    CL8
SRVCLASS DS    CL8
XCMD     DS    CL8Ø
CL2      DS    CL2
CL3      DS    CL3
CL4      DS    CL4
CL6      DS    CL6
RETC     DS    CL2
RSNC     DS    CL4
RETCODE  DS    F
RSNCODE  DS    F
RC       DS    F
WTOMSG   DS    ØF
WTOML    DS    H
WTOM     DS    CL8Ø
WTOMLEN  EQU   *-WTOM
WORKL    EQU   *-WORK
         REGISTER
         END
```

SAMPLE JCL TO CALL RESETWSO

```
//STEP1    EXEC PGM=RESETWSO,PARM='RESET,USERØØ1,TSOSRV'
//STEP2    EXEC PGM=RESETWSO,PARM='QUIESCE,BATCHØ1'
//STEP3    EXEC PGM=RESETWSO,PARM='RESUME,BATCHØ1'
//STEP4    EXEC PGM=RESETWSO,PARM='RESET,USERØØ1,TSOSRU'
```

JOBLOG GENERATED BY RESETWSO

```
IEF4Ø3I USERØØ2J - STARTED - TIME=17.35.21
+RESETWSO - COMMAND: RESET,USERØØ1,TSOSRV
+RESETWSO - FUNCTION: RESET   - ASNAME: USERØØ1  - SRVCLASS: TSOSRV
-                                          --TIMINGS (MINS.)--
-JOBNAME  STEPNAME PROCSTEP    RC   EXCP    CPU    SRB  ELAPS   SERV
-USERØØ2J STEP1                ØØ      3    .ØØ    .ØØ     .Ø    225
+RESETWSO - COMMAND: QUIESCE,BATCHØ1
+RESETWSO - FUNCTION: QUIESCE - ASNAME: BATCHØ1
+RESETWSO - ERROR - IWMRESET  RC = Ø4
-USERØØ2J STEP2                Ø8      5    .ØØ    .ØØ     .Ø    253
+RESETWSO - COMMAND: RESUME,BATCHØ1
+RESETWSO - FUNCTION: RESUME  - ASNAME: BATCHØ1
+RESETWSO - ERROR - IWMRESET  RC = Ø4
-USERØØ2J STEP3                Ø8      5    .ØØ    .ØØ     .Ø    262
+RESETWSO - COMMAND: RESET,USERØØ1,TSOSRU
+RESETWSO - FUNCTION: RESET   - ASNAME: USERØØ1  - SRVCLASS: TSOSRU
+RESETWSO - ERROR - IWMRESET  RC = ØC / RSN = ØC28
-USERØØ2J STEP4                Ø8      5    .ØØ    .ØØ     .Ø    267
IEF4Ø4I USERØØ2J - ENDED - TIME=17.35.21
```

The IWMRESET service has the same restrictions as the RESET system command:

- There are no restrictions for the RESET command when the originating and target service classes are both customer-defined.
- Attempts to move a privileged or high dispatching priority address space into a customer-defined service class are rejected.
- RESUME causes the work to be reclassified according to the service policy in effect and resumes processing at the performance targets specified in this service policy.

Systems Programmer
(France)

z/OS 64-bit real storage

OVERVIEW

IBM continues the development evolution, which started in 1970 with the introduction of S/370, with the introduction of the following: 64-bit real storage support in OS/390 Version 2 Release 10, z/Architecture and the IBM eServer zSeries 900, EXCP and EXCPVR with 64-bit IDAWs, 64-bit virtual memory capabilities for Assembler programmers in z/OS 1.2, AMODE 64 support extended to the Binder, and z/OS Loader in a future z/OS Release.

With z/Architecture, there have been significant changes at the hardware level with the introduction of up to five levels of Dynamic Address Translation tables for an effective 64-bit virtual address, as opposed to the two-level segment and page look-up process used with 31-bit. There is a new SIGP order, which will set the Architecture Mode – the Address Space Control Element

(ASCE), which resides in CR1, CR7, CR13, and the Address Space Second Table (ASTEs). The PSW is now 128 bits long. The low-address Prefixed Storage Area (PSA) has been completely reorganized and is now two 4KB-frames on an 8KB boundary. The Prefix Register is now 64 bits wide. There is also a host of new Assembler instructions and new instruction formats.

The IBM @SERVER zSERIES 900 provides the following Architecture modes:

- 24-bit addressing
- 31-bit addressing
- 64-bit addressing
- 64-bit GPRs
- 64-bit CRs
- 128-bit PSWs
- 15 exabyte addressing
- 32 and 64-bit arithmetic and logical operations
- 8KB PSA
- New instructions
- New format IDAWs
- New format Dynamic Address Translation structure.

With the Z900 Intelligent Resource Director we have support for:

- Dynamic Channel Path Management.

  This function allows the system to dynamically manage channel paths in response to changing workload demands. This is exclusive to z900 and z/OS.

- LPAR CPU Management.

  This function falls into two areas:

  - LPAR Weight Management, which dynamically manages a partitions CPU access based on workload demands and goals.
  - Vary Logical CPU Management, which optimizes the number of logical CPs based on a partition's current weight and CPU consumption.
- Channel System Priority Queueing.

  This function prioritizes I/O within an LPAR cluster. The LPAR priorities are based on workload goals. This is exclusive to z900.

OS/390 VERSION 2 RELEASE 10

Prior to OS/390 Version 2 Release 10, the ESA/390 architecture limited the amount of central storage that could be configured to a single OS/390 image to 2 gigabytes (2GB). With the introduction of OS/390 running on an IBM @SERVER Zseries 900 (launched in October 2000), up to 128GB of central storage can be supported when running in z/Architecture mode (also known as 'large real support', 64-bit real storage support, or ESA64). The maximum amount of real storage supported on a 64-bit capable processor is 256GB. The 128GB limit is a software restriction, which was introduced by the OS/390 designers to prevent the Page Frame Table from exceeding 1MB. This is a temporary software limitation.

Note: implementing OS/390 Version 2 Release 10 on a 9672 G5 or G6 processor does not enable the 64-bit real storage capability. OS/390 V2 R10 real storage layout is illustrated in Figure 1.

In a 64-bit real storage environment, the terms 'above the bar' and 'below the bar' are used to identify the areas between 2**31 and 2**64 -1, and 0 and 2**31-1 respectively. For example any address in the range 0 to 7FFFFFFF would be below the bar, and addresses in the range FFFFFFFF to 7FFFFFFF_FFFFFFFF would be above the bar.

64-bit real addressing can theoretically allow central storage to support 2**64 bytes of memory, which is 16 exabytes (8 billion times 2GB). In OS/390 Version 2 Release 10, the size of address spaces and data spaces is not increased beyond 2GB.

OS/390 Version 2 Release 10 can operate on both the current ESA/390 architecture and the new ESA64 architecture (refer to

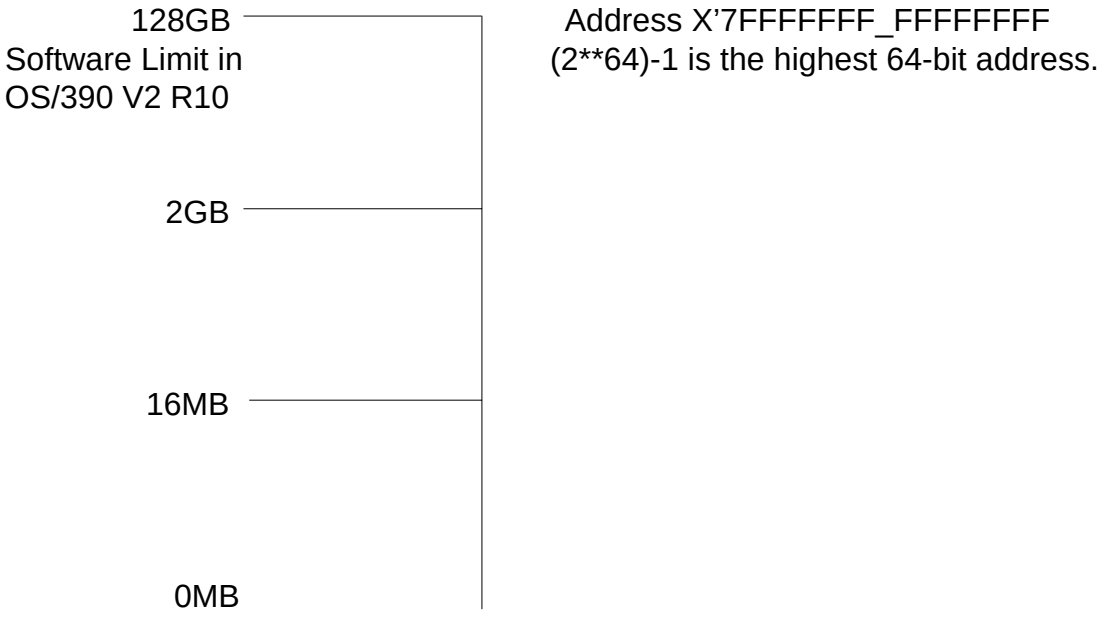

Figure 1: Real storage layout in OS/390 V2 R10

section *IPL/NIP*). When operating in ESA64 architecture, the following functions are no longer supported:

- Elimination of expanded storage.

  As the ratio of central storage to expanded storage decreases, the amount of overhead to move data into and out of expanded storage increases. To eliminate this overhead, there is a growing need to support all processor memory as

central storage, hence the move to 64-bit real storage support. Hiperspace services have been re-implemented to use real storage rather than expanded storage. The system use of expanded storage has also been re-implemented to use central storage. The MOVE PAGE instruction (MVPG) now works moving real-to-real. This change means that a page-out (DirectPO) SYSEVENT must decide whether a VIO or Standard Hiperspace page should be placed in central storage or sent to auxiliary storage when running in Z/Architecture mode, rather than the current process of deciding between expanded and auxiliary storage. As a direct consequence of this change, the Expanded Storage Table (EST) is no longer required.

- Deletion of Virtual Fetch.

  Installations can convert to using LLA for the IMS applications.

- Deletion of Duplexing Common and PLPA datasets.

- Deletion of SWAP dataset support.

  Installations must update the PAGTOTL parameter in IEASYS*xx* to no longer define the SWAP datasets. To compensate for this change, additional local page datasets can be added to restore the capacity. The PAGTOTL is now specified as follows in OS/390 Version 2 Release 10:

  ```
  PAGTOTL(ppp)
  ```

To use 64-bit real addressing support in OS/390 Version 2 Release 10, the following steps must be performed:

- You must install a zSeries 900 Processor.
- Configure all processor memory as real.
- Add the statement ARCHLVL 2 to LOAD*xx* (refer to section *IPL/NIP*).
- IPL OS/390 V2R10.

A new LOAD*xx* statement, ARCHLVL, has been introduced so that the IPL/NIP process prepares the system for initialization in either ESA/390 or z/Architecture mode:

```
ARCHLVL n
```

where n = 1 (for ESA/390) or n = 2 (for ESA64).

When the ARCHLVL statement in LOAD*xx* specifies ARCHLVL 1, the system IPLs in ESA/390 mode. When ARCHLVL 2 is specified, the system IPLs in z/Architecture mode.

The value corresponds to the appropriate nucleus-dependent code. The nucleus is spilt into three major parts:

- IEANUC0*x* – architecture independent
- IEANUC1*x* – ESA/390 dependent code
- IEANUC2*x* – z/Architecture dependent code

The architecture is dynamically chosen at initialization and the current nucleus parts are dynamically bound. A z/Architecture system always initializes to ESA/390 mode during IPL, at a later stage, an operating system causes a switch to z/Architecture mode by issuing a SIGP using a new Set-Architecture order. The SIGP will function correctly only if all other CPUs in the configuration are in the manual or check state (ie only one CPU is running).

IBM has added new reason codes for WAIT State 088:

- 0E – could not locate the nucleus extension.
- 10 – ESA64 nucleus extension was requested but the hardware does not support ESA64 architecture.

Below is an updated LOAD*xx* member with the new ARCHLVL parameter defined:

```
NUCLEUS   1
NUCLST    RP
```



```
ARCHLVL   2
IEASYM    (RP,L)
SYSPARM    RP
SYSCAT    MRMCAT113CCATALOG.MRMCAT1.MASTER
IODF  ØØ SYSB     RACB     ØØ
PARMLIB   SYS.SPROGS.PARMLIB
PARMLIB   SYS1.PARMLIB
*---------------- Definitions for SCØ1 -------------------------------
HWNAME    H1
LPARNAME  SCØ1
SYSCAT    SC1CAT113CCATALOG.SCØ1.MASTER
*---------------- Definitions for SCØ2 -------------------------------
HWNAME    H2
LPARNAME  SCØ2
SYSCAT    SC2CAT113CCATALOG.SCØ2.MASTER
```

The new architecture can be displayed with the D IPLINFO command as follows:

```
D IPLINFO

IEE254I  Ø7.13.31 IPLINFO DISPLAY 615
            SYSTEM IPLED AT 2Ø.45.16 ON 12/29/2ØØ1
          RELEASE OS/39Ø Ø2.1Ø.ØØ
          USED LOAD11 IN SYS1.IPLPARM ON Ø8ØØ
          ARCHLVL = 2   MTLSHARE = N
          IEASYM LIST = ØØ
          IEASYS LIST = ØØ
            IODF DEVICE Ø8ØØ
            IPL DEVICE Ø8Ø1 VOLUME MVRØØ1
```

DUAL CODE

As with previous operating system architectures, dual code must be implemented one for each architecture environment. To support dual code, a new flag CVTSAME (if set to one, indicates the presence of ESAME hardware) in CVTFLAG3 has been provided for execution time testing of which architecture is being used, although IBM documentation says that it is often simpler to check whether PSA field FLCARCH is non-zero. CVT bit CVTH6610 indicates whether OS/390 is at least at the Release 10 level.

Two new PSA macro definitions have been defined:

- IHAPSAE – PSA Extension (PSAE). The PSAE maps the ESAME format of the first page of the PSA.

- IHAPSAX – PSA Extension (PSAX). The PSAX maps the architected second page of the PSA.

SYSSTATE MACRO

The SYSSTATE macro has been enhanced to allow the caller to address the situation where macros may expand differently depending on the architecture being used. The ARCHLVL parameter has been added:

- ARCHLVL= 0 – the architecture is ESA/390.
- ARCHLVL=1 – the architecture is ESA/390 but includes the ESA/390 architecture items required by OS/390 R10 (eg the relative/immediate instructions).
- ARCHLVL= 2 – the architecture is z/Architecture. The macros that pay attention to the ARCHLVL will avoid generating z/Architecture instructions when the ARCHLVL < 2 is in effect.

RSM CHANGES Z/ARCHITECTURE MODE

There have been a number of changes to the Real Storage Manager (RSM) when running in z/Architecture mode.

- Non-fixed pages, including disabled reference (DREF), can be backed anywhere in real storage.
- VIO uses frames above the 2GB line.
- Nucleus, SQA, and LSQA (except DREF) will be backed below 2GB.
- The type of frame (above or below 2GB) used when a page is fixed is determined by an attribute specified at the time the virtual storage is obtained.
- Changes have been made to the DSPSERV macro interface to allow callers to specify whether pages of a data space can be backed anywhere in real storage when used for I/O (IOON fixed).

  - BACK=31 – specifies that the data space pages will be backed by central storage below the 2GB line when defined IOON.
  - BACK=64 – specifies that the data space pages will be backed by central storage above or below the 2GB line when defined IOON.
- Paging I/O is now performed directly to and from any real frame.

VSM CHANGES IN Z/ARCHITECTURE MODE

In OS/390 Version 2 Release 10 the GETMAIN, STORAGE OBTAIN, and CPOOL have new parameters to support real storage above 2GB.

Pageable pages are backed anywhere in central storage. The nucleus, fixed SQA, and fixed LSQA are always backed below the 2GB line in central storage. When a page is fixed (eg for I/O), it is fixed as requested by the second sub-parameter of LOC on the GETMAIN/STORAGE macro:

- LOC=(..,BELOW) or LOC=(..,24) – virtual storage below 16MB backed by central storage below 16MB when fixed.
- LOC=(..,ANY) or LOC=(..,31) – virtual storage below 2GB backed by central storage below 2GB when fixed.
- LOC=(24,31) – specifies virtual below 16MB, backed by central storage below 2GB when fixed.
- LOC=(24,64) – specifies virtual below 16MB, backed by central storage anywhere when fixed.
- LOC=(31,64) – specifies virtual below 2GB, backed by central storage anywhere when fixed.
- LOC=(..,64) – anywhere in central storage.

For data spaces, the option is BACK=31 or BACK=64.

Note: the old values of BELOW and ANY continue to be accepted and relate to 23 and 31, respectively.

Prior to OS/390 Version 2 Release 10, storage was obtained on a doubleword boundary, except when BNDRY=PAGE was specified. Two new parameters on the GETMAIN and STORAGE macros, STARTBDY (starting boundaries) and CONTBDY (containing boundaries), have been introduced:

- STARTBDY – the STARTBDY parameter indicates that the allocation must begin on a specified power of 2 alignment boundary. It is not restricted to certain subpools.
- CONTBDY – this parameter can be used to ensure that the allocated storage must not span a particular power of 2 alignment boundary, and must reside within the area identified by the containing boundary.

Note: STARTBDY and CONTBDY are not valid with LOC=EXPLICIT or BNDRY=PAGE.

Prior to OS/390 Version 2 Release 10, a cell pool managed by CPOOL was aligned on a doubleword boundary. In OS/390 Version 2 Release 10, CPOOL now supports alignment on a quadword boundary when requested by the BNDRY=QWORD parameter.

LRA INSTRUCTION

The LOAD Real Address instruction does not work when the real address is greater than 2GB because it places the result in a 32-bit general purpose register. Errors are detected by the system by generating a program check.

For the following fixed storage, which is backed below the 2GB line, the LRA instruction will work correctly:

- Nucleus, Fixed SQA, Fixed LSQA.
- Acquired with LOC=(…,24) or LOC=(…,31) or BACK=31.

The rule is to issue LRA only against areas that are fixed.

IBM recommends using the Test Protection Instruction (TPROT) instead of the LRA instruction when a program is using it to verify

that the virtual address is translatable and the page backing it is in real storage. For example, DB2 modules that use the LRA to determine whether a page resides in central storage have been changed to use TPROT.

PAGE FRAME TABLE

The Page Frame Table, which describes real storage, has been moved out of the Common Area. Removing the Page Frame Table from the system Common Area provides 16MB of virtual constraint relief for each 2GB of installed processor memory. For example, a system with 32GB of central storage will get back 256MB of virtual storage. An entry for the RSM Page Frame Table Data Space, used by the RSM to house the Page Frame Table, will now be added to the PASN access list of every address space. This makes the Page Frame Table Space globally addressable, similar to the Common Area Data Spaces.

SYNCHRONIZING TASKS USING PAUSE RELEASE AND TRANSFER

Although it's not directly related to 64-bit support, I would like to mention a new callable service that has been introduced recently, which allows you to synchronize tasks with minimal overhead. These new services, which are available to authorized and non-authorized callers in Assembler, use a system-managed object called a system-managed Pause Element (PE), which is similar to an ECB that is used for WAIT/POST and EVENTS processing. A pause element is used to pause and release a task and is used by the system to control whether or not a task is dispatchable. The system does not allow more than one dispatchable unit to be paused under a single PE.

The new service provides the following functions:

- Pause the current task (Pause service).
- Release a paused task (Release service).
- Simultaneously release a paused task and pass control to it (Transfer service).

- Simultaneously release one paused task and pass the current work unit (Transfer service).

There are five callable services available for task synchronization:

- Allocate_Pause_Element (IEAVAPE)
- Pause (IEAVPSE)
- Release (IEAVRLS)
- Transfer (IEAVXFR)
- Deallocate_Pause_Element (IEAVDPE).

A program must first allocate a PE by calling the Allocate_Pause_Element service. In response, the system allocates a PE and returns a pause element token (PET) that identifies the pause element (PE). The Pause, Release, and Transfer services can then be used. To return the PE to the system, the Deallocate_Pause_Element service must be issued. Though the PE remains allocated until you de-allocate it, you can use a PET for only one pair of calls, which results in a pause and a release of a task.

The Transfer Service provides new functions and a distinct advantage over WAIT and POST as follows:

- Release a paused task and transfer control directly to the released task.
- Pause the current task, release a paused task, and transfer control directly to the released task.

To invoke these callable services, the calling program object code must be linked with the linkable stub routine (IEACSS form SYS1.CSSLIB), or have the calling program LOAD and then call the service. The High Level Language (HLL) definition for the callable services is IEAASM, which is contained in SYS1.MACLIB.

EXCP AND EXCPVR CHANGES TO SUPPORT 64-BIT REAL STORAGE

Channel access to real storage above 2GB can be performed using the format-0 or format-1 CCWs, in conjunction with a new

64-bit indirect-data-address word (IDAW). The new 64-bit IDAW is also known as a Format 2 IDAW. Sixty-four-bit IDAWS are important because they allow I/O to be done using real storage above 2GB, which eliminates the need for the system to move the data into storage below 2GB before performing the I/O. The movement of data can be very costly in terms of CPU overhead.

The 64-bit IDAW has the following attributes:

- 8 bytes in length
- Must be aligned on a doubleword boundary
- Can have a span of either 2KB or 4KB. This is a change from 31-bit IDAWs, which could process only up to 2KB of real storage.

A flag in the Operation Request Block (ORB) indicates which format of IDAW is being used. A flag in the ORB indicates whether the span of an IDAW is to be 2KB or 4KB.

IOS modifies the UCB to indicate whether 64-bit IDAWs are supported for a specific unit. If an I/O request specifies a 64-bit IDAW, but the UCB indicates that 64-bit IDAWs are not supported, IOS will post an error.

A program can determine whether 64-bit IDAWs are supported by looking at bit UCBEIDAW in the UCB for the device:

```
TM   UCBFL7,UCBEIDAW
```

Does the device support 64-bit IDAWs?

```
JZ     REAL31
```

No – UCBEIDAW is set only when the system is running in z/Architecture mode and the device support code (UIM) supports them.

IOS support for 64-bit real storage can be summarized as follows:

- 64-bit IDAWs.
- The STARTIO interface supports the use of channel programs that use the 64-bit IDAWs.

- DASD and TAPE are supported for I/O involved with real storage above 2GB.
- Through the UIM definition, IOS is aware of which device types support the 64-bit IDAW and updates the UCB accordingly.

For applications that invoke EXCP directly, no changes are required to exploit 64-bit IDAWs, since the channel program provided by the application is always translated into a real channel program. To request storage for I/O buffers that can be fixed in real storage above 2GB, you must change the application. All that is required for DASD and TAPE applications is a change to the LOC specification when the I/O buffer(s) are obtained using the GETMAIN or STORAGE OBTAIN services (refer to section *VSM changes in z/Architecture mode*). These services have been changed so that storage can be fixed in real storage above 2GB using the following specification:

```
LOC=(…,64)
```

EXCPVR

It is the responsibility of programs that use EXCPVR to page fix all I/O areas and to build real channel programs. All CCW chains must be changed to exploit I/O buffers backed in real storage above 2GB (the channel program must reside below 16MB). In addition to changing the GETMAIN or STORAGE OBTAIN request, they must build a 64-bit IDAW to replace the 31-bit IDAW used in ESA-390 mode and build an IOBE to specify that 64-bit IDAWs are being passed. IBM has defined a new flag in the IOBE (IOB Extension) for this purpose:

- IOBEEIDAW – using 64-bit IDAWs (UCBEIDAW= '1'B)

Consider the following when using 64-bit IDAWs:

- All CCWs in a CCW chain that specify the use of an IDAW must use the same format IDAW.
- Each 64-bit IDAW must begin on a doubleword boundary.

- Each 64-bit IDAW can process up to 4KB of real storage (this is a change from 31-bit IDAWs, which can process only up to 2KB of real storage).
- If a program is to use 64-bit IDAWs, but does not currently create an IOB extension, it must first be changed to create an IOBE, and set flag IOBCEF in the IOB indicating that an IOB extension is being used. The address of the IOBE is passed to EXCPVR in Register 0. The IOBE is mapped by the IOSDIOBE macro.
- The 64-bit real address of the storage can be stored in the IDAW using the new instruction Store-Real-Address, or STRAG. This new instruction has been introduced to aid in channel program construction, which does not require the use of 64-bit registers. The STRAG instruction accepts a virtual address as an argument and then stores the translated 64-bit real address at an 8-byte virtual storage location specified as the other argument.

Format: STRAG D1(B1),D2(B2) or STRAG IDAW,IOBUFFER

STRAG does not set the condition code. If the translation is not available, the processor raises a translation exception (PIC X'10', X'11' etc). Therefore, it cannot be used to determine whether a page is currently backed by a real frame.

EXCPVR AND FORMAT-1 CCWs

Starting with OS/390 Version 2 Release 10, an EXCPVR application can build and execute Format-1 CCW in ESA/390 and z/Architecture mode. Format-1 CCWs use 31-bit addressing as opposed to Format-0 CCWs, which can use only 24-bit addresses. Format-1 CCWs are utilized by components such as the Auxiliary Storage Manager (ASM) for paging I/O, VSAM, and GTF. Format-1 chains are requested by setting bit IOBEFMT1 in the IOBE. APAR OW46454 extends the Format-1 CCW support to allow CCW chains to be located anywhere in virtual storage.

ACCESS METHOD CHANGES

The following access methods have been changed to utilize large real storage. This requires no application changes to take advantage of large real storage except if the application acquires the buffers, in which case it is responsible for obtaining storage with the correct attributes:

- SAM, BSAM, and QSAM but only for DASD and TAPE support.
- VSAM, which provides support for Extended Format datasets.

R F Perretta
Systems Consultant
Millenium Computer Consultancy (UK)

An EDIT macro that you always want to have

Here is a small ISPF EDIT/BROWSE macro, called CURS, which will show the dataset under the cursor.

It is often necessary to look into a job or some parameter member and follow the references to other datasets. With this small macro you can follow the reference.

The macro is very simple. From the current cursor position, it tries to scan forward and backward to find a string like a dataset name.

The macro could be improved or extended with new functions such as:

- Display dataset information.
- Add the dataset to a reference list.

You can add the macro to the EDIT/BROWSE keylist.

Just put your cursor under the dataset name, press PF13, and see the dataset.

THE CODE

```
/*    REXX                                                        */
/* A small REXX EXEC to EDIT or BRowse the dataset under cursor */
/* You can assign a PFKEY via "KEYS" ISPF command :             */
/* PF13    ECUR EDIT                                            */
/* PF14    ECUR BROWSE                                          */
/*                                                              */

address ispexec
'control errors return'
address isredit
'macro ( cmd  ) '

parse upper var cmd command                    /* make to upper case */

if command = '' then command = 'EDIT'          /*default is EDIT     */
else
  if command <> 'EDIT' & command <> 'BROWSE'  then
  do
    say 'Invalid command ' command
    return
  end

, (row,col) = cursor'                          /* get cursor position */
'('vline') = line ' row

/* scan forward from the current cursor position to find the start of
   the dataset name accept all possible vaild dsname characters     */
i = col
cont = 1
do while(i >  1 & cont = 1)
  c =  substr(vline,i,1)
  if  datatype(c,'a') = 1 | datatype(c,'n') = 1 | c = '.' |,
      c = ')' | c = '(' |,
      c = '#' | c = '$' | c = 'õ', /*national chars ? */
  then
    i = i -1
  else
      cont = Ø
end

/* now scan backward                                                */
si = i+1
cont = 1
i = col
do while(i < 8Ø & cont = 1)
  c =  substr(vline,i,1)
  if  datatype(c,'a') = 1 | datatype(c,'n') = 1 | c = '.' |,
      c = ')' | c = '(' |,
```

```
      c = '#' | c = '$' | c = 'õ', /*national chars */
  then
      i = i+1
  else
      cont = Ø
end

/* if it is "(dsn)" drop parents                                    */
if substr(vline,si,1) = '(' & substr(vline,i-1,1) = ')' then
 do
   si = si+1
   i  = i-1
end

/* search for the first alpaha chars                                */
cont = 1
do while(si < 8Ø & cont = 1)
  if datatype(substr(vline,si,1),'a') = 1 then
    cont = Ø
  else
    si = si+1
end

/* now we have it , check if it is realy a dsn                     */
dsn = substr(vline,si,i-si)
if listdsi(dsn) <> Ø then
  say "INVALID DSN "dsn" UNDER THE CURSOR"
else
do
  address ispexec
  command "dataset('"dsn"')"
end
return
```

Miklos Szigetvari
System Programmer (Austria)

Sorting storage tables

One of the common requirements in programming Assembler programs is to invoke a routine to sort an in-storage table. A simple ‘bubble sort’ works well for small tables, but the performance degrades drastically as the number of rows increases.

The following code presents a general purpose ‘shell sort’ program that offers significant performance improvements over the ‘bubble sort’ for larger tables.

An Assembler macro, MEMSORT, is provided to make the construction of the parameter list for the Assembler routine easier and more readable.

STANDARD AND EXECUTE FORM SYNTAX

The standard and execute forms of the MEMSORT macro are written as follows:

- name – name: symbol. Begin name in column 1.
- MEMSORT – one or more blanks must precede MEMSORT. One or more blanks must follow MEMSORT.
- AREA=area_address – RX-type address or register (2) - (12).
- ,AREALEN=area_len – RX-type address or register (2) - (12).
- ,RECLEN=record_length – literal value or register (2) - (12).
- ,FIELDOFF=field_offset – literal value or register (2) - (12).
- ,FIELDLEN=field_length – literal value or register (2) - (12).
- ,DATATYPE=CHAR – default DATATYPE=CHAR
- ,DATATYPE=HEX
- ,DIR=A – default DIR=A
- ,DIR=D
- ,MF=S – standard form
- ,MF=(E,label) – execute form.

STANDARD AND EXECUTE FORM PARAMETERS

The parameters are explained as follows:

- AREA – the name (RX-type) or address in register (2)-(12) of the start of the storage table to be sorted.
- AREALEN – the name (RX-type) or address in register (2)-(12) of the fullword length of the storage table to be sorted.
- RECLEN – a literal or value in register (2)-(12) of the length of each logical record within the storage table. Valid range = 1 - 255. If a register is used, the rightmost byte is assumed to contain the value and the other three bytes are ignored.
- FIELDOFF – a literal or value in register (2)-(12) of the offset within the logical record of the sort field. Valid range = 0 - 254. If a register is used, the rightmost byte is assumed to contain the value and the other three bytes are ignored.
- FIELDLEN – a literal or value in register (2)-(12) of the length of the field that is to be used to sort the storage table. Valid range = 1 - 255. If a register is used, the rightmost byte is assumed to contain the value and the other three bytes are ignored.
- ,DATATYPE=CHAR ,DATATYPE=HEX – specifies how the data is to be treated by the sort program:
  - DATATYPE=CHAR – the characters ‘A’ through ‘F’ are treated as lower than ‘0’.
  - DATATYPE=HEX – the characters ‘A’ through ‘F’ are treated as higher than ‘0’.
- ,DIR=A ,DIR=D – specifies the direction of the sort:
  - DIR=A – the table is sorted in ascending sequence.
  - DIR=D – the table is sorted in descending sequence.
- ,MF=S – specifies the standard form of MEMSORT. The standard form places the parameters into an in-line parameter list.
- ,MF=(E,label) – specifies the execute form of MEMSORT. The execute form generates code to put the parameters into the storage pointed to by ‘label’.

LIST FORM SYNTAX

The list form of the MEMSORT macro is written as follows:

- name – name: symbol. Begin name in column 1.
- MEMSORT – one or more blanks must precede MEMSORT. One or more blanks must follow MEMSORT.
- MF=(L,label) – list form.

LIST FORM PARAMETERS

The parameters are explained as follows:

- ,MF=(L,label) – specifies the list form of MEMSORT. The list form generates code to reserve enough storage to contain the parameter list and assigns 'label' as the reference name.

EXAMPLES OF USING THE MEMSORT MACRO

Sort table TABLE whose size is in TABLEL in descending sequence. The sort field is 16 bytes long and is at offset 3 from the start of the 80-byte logical record. The sort field is character data.

```
          MEMSORT AREA=TABLE,                                    X
                AREALEN=TABLEL,                                  X
                RECLEN=80,                                       X
                FIELDOFF=3,                                      X
                FIELDLEN=16,                                     X
                DATATYPE=CHAR,                                   X
                DIR=D
          ...
          ...
TABLEL    DC    F'8000'              * Size of the table
TABLE     DS    100CL80              * Table
```

Sort table whose address is in R7 and length is in R8. The length of the logical record is stored in field RLEN. The sort field is at the start of the record and the whole record length should be used to sort the record in ascending sequence. The first four bytes of the record contain hex device numbers. This is a RENT program.

```
          XR    R6,R6
          ICM   R6,B'0001',RLEN
          MEMSORT AREA=(R7),                                          X
                AREALEN=(R8),                                         X
                RECLEN=(R6),                                          X
                FIELDOFF=0,                                           X
                FIELDLEN=(R6),                                        X
                DATATYPE=HEX,                                         X
                DIR=A,                                                X
                MF=(E,MEMSORT1)
          ...
          ...
WORKAREA  DSECT
SAVEAREA  DS    18F
RLEN      DS    X                    * Record length
          ...
          MEMSORT MF=(L,MEMSORT1)
```

SOURCE CODE FOR THE MEMSORT MACRO

```
        MACRO
.*---------------------------------------------------------------------
.* MACRO NAME : MEMSORT
.*
.* FUNCTION   : TO CONSTRUCT THE PARAMETER LIST FOR THE RDSSORT
.*            : ASSEMBLER PROGRAM THAT SORTS STORAGE TABLES
.*
.*
.* SYNTAX     : MEMSORT AREA=area_address,
.*                      AREALEN=area_length,
.*                      RECLEN=record_length,
.*                      FIELDOFF=field_offset,
.*                      FIELDLEN=field_length,
.*                      DATATYPE=CHAR/HEX,
.*                      DIR=A/D,
.*                      MF=S
.*                      MF=(L,label)
.*                      MF=(E,label)
.*
.* KEYWORDS   : AREA=area_address
.*                specifies the name (RX-type), or address in
.*                register (2)-(12), of the storage table to be
.*                sorted.
.*
.*              AREALEN=area_length
.*                specifies the name (RX-type), or address in
.*                register (2)-(12), of the fullword length of
.*                the area to be sorted.
```

```
.*
.*               RECLEN=record_length
.*                 specifies the length (1-255) of each logical
.*                 record within the storage table.
.*                 If a register is used to specify the length, the
.*                 rightmost byte is assumed to contain the value.
.*
.*               FIELDOFF=field_offset
.*                 specifies the offset (Ø-254) into the logical
.*                 record of the field to be sorted.
.*                 If a register is used to specify the offset, the
.*                 rightmost byte is assumed to contain the value.
.*
.*               FIELDLEN=field_len
.*                 specifies the length (1-255) of the field that is
.*                 used to sort the records.
.*                 If a register is used to specify the length, the
.*                 rightmost byte is assumed to contain the value.
.*
.*               DIR=A/D
.*                 specifies the direction of the sort (A=Ascending,
.*                 D=Descending).
.*
.*               DATATYPE=HEX/CHAR
.*                 specifies the type of data to be sorted.
.*
.*                   For DATATYPE=CHAR, characters A through F are
.*                   deemed to be lower in value than Ø.
.*
.*                   For DATATYPE=HEX, characters A through F are
.*                   deemed to be higher in value than Ø.
.*
.*             : MF=S
.*                 specifies the standard form of the macro.  The "S"
.*                 form generates code to put the parameters into an
.*                 in-line parameter list and invoke the desired
.*                 service
.*
.*             : MF=(L,label)
.*                 specifies the list form of the macro.  The "L" form
.*                 defines an area to be used for the parameter list.
.*                 All keywords applicable to the MEMSORT function
.*                 specified must be coded in order for the macro to
.*                 calculate the space required for the parameter list.
.*
.*             : MF=(E,label)
.*                 specifies the execute form of the macro. The "E" form
.*                 generates code to put the parameters into the storage
.*                 pointed to by 'label'.
```

```
.*
.*-------------------------------------------------------------------
&LABEL    MEMSORT &AREA=,                                            X
               &AREALEN=,                                            x
               &RECLEN=,                                             x
               &FIELDOFF=,                                           x
               &FIELDLEN=,                                           x
               &DIR=A,                                               x
               &DATATYPE=CHAR,                                       x
               &MF=S
.*-------------------------------------------------------------------
.* CHECK MACRO FORM
.*-------------------------------------------------------------------
&NUMMF    SETA  N'&MF
          AIF   ('&MF(1)' EQ 'L').CHKMFL
          AIF   ('&MF(1)' EQ 'S').INLINE
          AIF   ('&MF(1)' EQ 'E').USEMFE
          AGO   .ERROR1
.INLINE   CNOP  Ø,4
&LABEL    B     B&SYSNDX            Branch around parm list
A&SYSNDX DC    4F'Ø'               Inline Parameter List
B&SYSNDX LA    1,A&SYSNDX          Point to parm list
          AGO   .GETPARMS
.USEMFE   ANOP
          AIF   (&NUMMF NE 2).ERROR2
&MFLABEL  SETC  '&MF(2)'
          AIF  ('&MFLABEL'(1,1) EQ '(').MFREG
&LABEL   LA    1,&MFLABEL          Point to parameter list
          AGO   .GETPARMS
.MFREG    ANOP
&REG      SETC  '&MFLABEL'(2,K'&MFLABEL-2)
         LR    1,&REG              Point to parameter list
.GETPARMS ANOP
          AIF   ('&AREA' EQ '').ERROR3
          AIF   ('&AREALEN' EQ '').ERROR4
          AIF   ('&RECLEN' EQ '').ERROR5
          AIF   ('&FIELDOFF' EQ '').ERROR6
          AIF   ('&FIELDLEN' EQ '').ERROR7
          AIF   ('&DIR' EQ '').ERROR8
          AIF   ('&DATATYPE' EQ '').ERROR9
.*-------------------------------------------------------------------
.* PROCESS THE AREA KEYWORD
.*-------------------------------------------------------------------
.DOAREA   ANOP
          AIF  ('&AREA'(1,1) EQ '(').AREAREG
         LA    15,&AREA            Point to area
          AGO   .GOTAREA
.AREAREG  ANOP
&REG      SETC  '&AREA'(2,K'&AREA-2)
```

```
         LR    15,&REG             Point to parameter list
.GOTAREA  ANOP
         STCM  15,B'1111',Ø(1)     Store in parm list
.*
.*-------------------------------------------------------------------
.* PROCESS THE AREALEN KEYWORD
.*-------------------------------------------------------------------
.DOAREAL  ANOP
          AIF  ('&AREALEN'(1,1) EQ '(').AREALREG
         LA    15,&AREALEN         Point to area
          AGO   .GOTAREAL
.AREALREG ANOP
&REG      SETC  '&AREALEN'(2,K'&AREALEN-2)
         LR    15,&REG             Point to parameter list
.GOTAREAL ANOP
         STCM  15,B'1111',4(1)     Store in parm list
.*
.*-------------------------------------------------------------------
.* PROCESS THE RECLEN KEYWORD
.*-------------------------------------------------------------------
.DORECL   ANOP
          AIF  ('&RECLEN'(1,1) EQ '(').RECLREG
          AIF  ('&RECLEN' LT '1').ERROR11
          AIF  ('&RECLEN' GT '255').ERROR11
         MVI   12(1),&RECLEN        Move in record length
          AGO   .DOFIELDO
.RECLREG  ANOP
&REG      SETC  '&RECLEN'(2,K'&RECLEN-2)
         STCM  &REG,B'ØØØ1',12(1)
.*
.*
.*-------------------------------------------------------------------
.* PROCESS THE FIELDOFF KEYWORD
.*-------------------------------------------------------------------
.DOFIELDO ANOP
          AIF  ('&FIELDOFF'(1,1) EQ '(').FLDOREG
          AIF  ('&FIELDOFF' GT '254').ERROR12
         MVI   13(1),&FIELDOFF      Move in field offset
          AGO   .DOFIELDL
.FLDOREG  ANOP
&REG      SETC  '&FIELDOFF'(2,K'&FIELDOFF-2)
         STCM  &REG,B'ØØØ1',13(1)
.*
.*-------------------------------------------------------------------
.* PROCESS THE FIELDLEN KEYWORD
.*-------------------------------------------------------------------
.DOFIELDL ANOP
          AIF  ('&FIELDLEN'(1,1) EQ '(').FLDLREG
          AIF  ('&FIELDLEN' LT '1').ERROR13
```

```
          AIF  ('&FIELDLEN' GT '255').ERROR13
         MVI   14(1),&FIELDLEN       Move in field length
          AGO   .DODATA
.FLDLREG  ANOP
&REG      SETC  '&FIELDLEN'(2,K'&FIELDLEN-2)
         STCM  &REG,B'ØØØ1',14(1)
.*
.*-------------------------------------------------------------------
.* PROCESS THE DATATYPE KEYWORD
.*-------------------------------------------------------------------
.DODATA   ANOP
          AIF  ('&DATATYPE' EQ 'CHAR').DATACHAR
          AIF  ('&DATATYPE' EQ 'HEX').DATAHEX
          AGO  .ERROR14
.DATACHAR ANOP
          AIF  ('&DIR' EQ 'A').CHARA
          AIF  ('&DIR' EQ 'D').CHARD
          AGO  .ERROR15
.CHARA    MVI   15(1),C'A'           Ascending CHAR
          AGO  .ENDSYSIN
.CHARD    MVI   15(1),C'D'           Descending CHAR
          AGO  .ENDSYSIN
.DATAHEX  ANOP
          AIF  ('&DIR' EQ 'A').HEXA
          AIF  ('&DIR' EQ 'D').HEXD
          AGO  .ERROR15
.HEXA     MVI   15(1),C'B'           Ascending HEX
          AGO  .ENDSYSIN
.HEXD     MVI   15(1),C'E'           Descending HEX
          AGO  .ENDSYSIN
.*
.ENDSYSIN ANOP
         LA    15,12(1)             Point to SYSIN
         STCM  15,B'1111',8(1)      Store as PARM
.CALLSORT ANOP
         LINK  EP=RDSSORT           Call RDSSORT
          AGO   .END
.CHKMFL   ANOP
          AIF   (&NUMMF NE 2).ERROR1Ø
&MFLABEL  SETC  '&MF(2)'
&MFLABEL DS    ØF
         DS    F                    Address of Storage Table
         DS    F                    Address of Storage Table Length
         DS    F                    Address of Sort Parms
         DS    ØCL4                 Sort Parms
         DS    C                    --> Record Length
         DS    C                    --> Field Offset
         DS    C                    --> Field Length
         DS    C                    --> Sort Direction
```

```
         AGO   .END
.*-------------------------------------------------------------------
.* Error messages
.*-------------------------------------------------------------------
.ERROR1   MNOTE 12,'Invalid macro form'
         AGO   .END
.ERROR2   MNOTE 12,'Invalid EXECUTE form - Use MF=(E,label)'
         AGO   .END
.ERROR3   MNOTE 12,'AREA Not Specified'
         AGO   .END
.ERROR4   MNOTE 12,'AREALEN Not Specified'
         AGO   .END
.ERROR5   MNOTE 12,'RECLEN Not Specified'
         AGO   .END
.ERROR6   MNOTE 12,'FIELDOFF Not Specified'
         AGO   .END
.ERROR7   MNOTE 12,'FIELDLEN Not Specified'
         AGO   .END
.ERROR8   MNOTE 12,'DIR Not Specified'
         AGO   .END
.ERROR9   MNOTE 12,'DATATYPE Not Specified'
         AGO   .END
.ERROR1Ø  MNOTE 12,'Invalid LIST form - Use MF=(L,label)'
         AGO   .END
.ERROR11  MNOTE 12,'RECLEN is invalid - must be between 1 and 255'
         AGO   .END
.ERROR12  MNOTE 12,'FIELDOFF is invalid - must be between Ø and 254'
         AGO   .END
.ERROR13  MNOTE 12,'FIELDLEN is invalid - must be between 1 and 255'
         AGO   .END
.ERROR14  MNOTE 12,'DATATYPE is invalid - must be CHAR or HEX'
         AGO   .END
.ERROR15  MNOTE 12,'DIR is invalid - must be A or D'
         AGO   .END
.END     MEND
```

SOURCE CODE FOR THE RDSSORT PROGRAM

```
RDSSORT  TITLE 'PROGRAM TO SORT STORAGE TABLE'
*-------------------------------------------------------------*
* Name            : RDSSORT
*
* Function        : General purpose SORT function
*
* Attributes      : Amode(31)
*                   Rmode(Any)
*                   RENT
*
```

```
* Register Usage  :
*
* r1  - parameter passed : +Ø --> Address of SORTIN
*                        : +4 --> Address of Length of SORTIN
*                        : +8 --> Address of SYSIN
*
*                                 Format of SYSIN Bytes :
*                                   Record_length  DS  X
*                                   Field_offset   DS  X
*                                   Field_length   DS  X
*                                   Sort Direction DS  X :
*
*                                                  A = ASCEND CHAR
*                                                  B = ASCEND HEX
*                                                  D = DESCEND CHAR
*                                                  E = DESCEND HEX
*
* r2  - copy of parms/address of CLC to be performed
* r3  - address of SORTIN
* r4  - address of SORTIN length
* r5  - address of SYSIN
* r6  - address of field in record
* r7  - address of record
* r8  - address of record
* r9  - length of field
* r1Ø -
* r11 - address of field in temporary record
* r12 - base
* r13 - work area
*
*-------------------------------------------------------------------*
* This program performs general purpose sorting on information in
* storage.
*
* The SHELL SORT algorithm is used to provide much higher
* performance than bubble sort.
*
* The algorithm is as follows : (REXX)
*
* a) Data to be sorted in array 'Stem.'
*
* b) Generate the highest interval 'highint' :
*
*      highint = 1
*      do while (highint <= Stem.Ø/9)
*          highint= highint*3+1
*          end
*
* c) For each value of highint, perform insertion sort so that we have
*    a progressively smaller series of 'highint sorted' subsets in the
```

```
*    array :
*
*      do while (highint > Ø)
*        do i = highint+1 to Stem.Ø
*          temprec = Stem.i
*          j = i
*          k = j - highint
*          do while (j > highint) & (Stem.k < temprec)
*            Stem.j = Stem.k
*            j = j - highint
*            k = j - highint
*          end
*          Stem.j = temprec
*        end
*        highint = highint % 3
*      end
*
* d) Data in 'Stem.' is now sorted
*
* The following Assembler uses field offset and length values to
* enable substring sorting (REXX above just sorts the records).
* It also handles ascending/descending sorts by pointing R2 at
* the executed compare.
*
*-------------------------------------------------------------------*
RDSSORT  CSECT
RDSSORT  AMODE 31
RDSSORT  RMODE ANY
         BAKR  R14,RØ                linkage stack
         LR    R12,R15               copy entry address to base
         USING RDSSORT,R12           address it
         MODID
         LR    R2,R1                 copy parms passed
GETSTOR  EQU   *
         STORAGE OBTAIN,             get the workarea storage        X
               LENGTH=WORKLEN,       this much                       X
               ADDR=(R13),           put address in r13              X
               SP=Ø,KEY=8,           subpool Ø storage key 8         X
               LOC=ANY,              above the line                  X
               COND=NO               unconditional
         USING WORKAREA,R13          address workarea
         LR    R14,R13               Copy workarea address
         L     R15,=A(WORKLEN)       Get length of workarea
         XR    RØ,RØ                 Clear
         XR    R1,R1                 Clear
         MVCL  R14,RØ                Init to zeros
         MVC   4(4,R13),=C'F1SA'     set acronym
GETPARMS EQU   *
         LM    R3,R5,Ø(R2)           Get the passed parms
```

```
         XR    R1,R1                Clear
         ICM   R1,B'ØØØ1',Ø(R5)     Get the record length
         BZ    BADSORT              If zero - error
         STCM  R1,B'1111',RECLEN    Store as fullword
         XR    R1,R1                Clear
         IC    R1,1(R5)             Get the field offset
         STCM  R1,B'1111',FLDOFF    Store as fullword
         XR    R1,R1                Clear
         IC    R1,2(R5)             Get the field length
         STCM  R1,B'1111',FLDLEN    Store as fullword
         A     R1,FLDOFF            Re-add offset
         C     R1,RECLEN            Compare to record length
         BH    BADSORT              Bigger - therefore error
         ICM   R1,B'1111',Ø(R4)     Get the SORTIN length
         BZ    BADSORT              if zero - error
         XR    RØ,RØ                Clear
         D     RØ,RECLEN            Divide by record length
         STCM  R1,B'1111',NUMRECS   Store as number of records
         LTR   R1,R1                Test for records
         BZ    BADSORT              if zero - error
         XR    RØ,RØ                Clear
         ICM   R1,B'1111',NUMRECS   Get number of records
         D     RØ,=F'9'             Calculate interval index
         STCM  R1,B'1111',INTINDX   Store
         LA    R2,ASCEND            Assume ascending
         CLI   3(R5),C'D'           Is it descending ?
         BE    SETDCEND             Yes
         CLI   3(R5),C'E'           Is it descending ?
         BE    SETDCEND             Yes
         B     GETINTVL             Leave as ascending
SETDCEND EQU   *
         LA    R2,DESCEND           Set to descending
GETINTVL EQU   *
         MVC   HIGHINT,=F'1'        Set high interval to 1
         CLI   3(R5),C'A'           CHAR?
         BE    INTVLOOP             Yes
         CLI   3(R5),C'D'           CHAR?
         BE    INTVLOOP             Yes
         OI    SORTTYPE,ISHEX       Indicate
INTVLOOP EQU   *
         CLC   HIGHINT,INTINDX      Compare interval against index
         BH    GOTINTVL             If higher got High Interval
         XR    RØ,RØ                Clear
         ICM   R1,B'1111',HIGHINT   Get the current setting
         M     RØ,=F'3'             multiply by 3
         LA    R1,1(R1)             and add 1
         STCM  R1,B'1111',HIGHINT   Store it
         B     INTVLOOP             And loop round
GOTINTVL EQU   *
```

```
         CLC   HIGHINT,=F'Ø'          Is high interval > Ø ?
         BNH   SORTED                 no - sorted
         ICM   R1,B'1111',HIGHINT     Get high interval
         LA    R1,1(R1)               Add one
         ST    R1,IINDEX              Store it
DOINTVL  EQU   *
         CLC   IINDEX,NUMRECS         Compare to number of records
         BH    NEXTINT                if > num recs then redo interval
         LR    R7,R3                  get start address
         XR    RØ,RØ                  Clear
         M     RØ,RECLEN              Multiply by record length
         S     R1,RECLEN              adjust index
         AR    R7,R1                  Add to start = record
         L     R15,RECLEN             Get record length
         BCTR  R15,RØ                 Minus 1 for move
         EX    R15,MOVETEMP           Copy record to temporary store
         MVC   JINDEX,IINDEX          Copy index
         ICM   R1,B'1111',JINDEX      Load replace index
         S     R1,HIGHINT             Subtract interval
         STCM  R1,B'1111',KINDEX      Store as temporary index
DOSUBSET EQU   *
         CLC   JINDEX,HIGHINT         Compare to interval
         BNH   ENDSUBS                if not higher - end
         LR    R7,R3                  copy address of SORTIN
         XR    RØ,RØ                  Clear
         ICM   R1,B'1111',KINDEX      Get temp index
         M     RØ,RECLEN              Multiply by rec length
         S     R1,RECLEN              adjust index
         AR    R7,R1                  Add to start = record
*
         XC    R6REC(255),R6REC       Clear
         XC    R11REC(255),R11REC     Clear
         XR    R15,R15                Clear
         L     R15,RECLEN             Get length of record
         BCTR  R15,RØ                 -1
         LA    R11,TEMPREC            Point to temporary record
         EX    R15,MOVER11            Copy in to workarea
*
         LA    R11,R11REC             Point to temporary record
         TM    SORTTYPE,ISHEX         HEX ?
         BNO   NOTRAN11               No - no translation
         TR    R11REC(255),TABLE
*
NOTRAN11 EQU   *
         A     R11,FLDOFF             Add the offset
         LR    R6,R7                  Point to record
         XR    R15,R15                Clear
         L     R15,RECLEN             Get length of record
         BCTR  R15,RØ                 -1
```

```
         EX    R15,MOVER6            Copy in to workarea
         TM    SORTTYPE,ISHEX        HEX ?
         BNO   NOTRAN6               No - no translation
         TR    R6REC(255),TABLE
NOTRAN6  EQU   *
         LA    R6,R6REC              Point to temporary record
         A     R6,FLDOFF             Add the offset
         XR    R9,R9                 Clear
         L     R9,FLDLEN             Load length of field
         BCTR  R9,RØ                 Minus 1 for CLC
         EX    R9,Ø(R2)              Perform the CLC
         BNH   ENDSUBS               if not higher - end
SWAPRECS EQU   *
         LR    R7,R3                 copy address of SORTIN
         XR    RØ,RØ                 Clear
         ICM   R1,B'1111',JINDEX     Get replace index
         M     RØ,RECLEN             Multiply by rec length
         S     R1,RECLEN             adjust index
         AR    R7,R1                 Add to start = record
         LR    R8,R3                 copy address of SORTIN
         XR    RØ,RØ                 Clear
         ICM   R1,B'1111',KINDEX     Get replace index
         M     RØ,RECLEN             Multiply by rec length
         S     R1,RECLEN             adjust index
         AR    R8,R1                 Add to start = record
         L     R15,RECLEN            Get record length
         BCTR  R15,RØ                Minus 1 for move
         EX    R15,MOVESWAP          Swap the records
         ICM   R1,B'1111',JINDEX     Load up j index
         S     R1,HIGHINT            Subtract high interval
         STCM  R1,B'1111',JINDEX     Store again
         ICM   R1,B'1111',JINDEX     Load up j index
         S     R1,HIGHINT            Subtract high interval
         STCM  R1,B'1111',KINDEX     Store again
         B     DOSUBSET
ENDSUBS  EQU   *
         LR    R7,R3                 copy address of SORTIN
         XR    RØ,RØ                 Clear
         ICM   R1,B'1111',JINDEX     Get replace index
         M     RØ,RECLEN             Multiply by rec length
         S     R1,RECLEN             adjust index
         AR    R7,R1                 Add to start = record
         L     R15,RECLEN            Get record length
         BCTR  R15,RØ                Minus 1 for move
         EX    R15,MOVEBACK          Copy back the temp record
THISNEXT EQU   *
         ICM   R1,B'1111',IINDEX     Get value for this index
         LA    R1,1(R1)              Add one
         STCM  R1,B'1111',IINDEX     Store as high index
```

```
         B     DOINTVL                Loop round
NEXTINT  EQU   *
         ICM   R1,B'1111',HIGHINT     Get the high interval
         XR    RØ,RØ                  Clear
         D     RØ,=F'3'               Divide by three
         STCM  R1,B'1111',HIGHINT     Store
         B     GOTINTVL               and loop round
BADSORT  EQU   *
         LA    R2,8                   Set rc
SORTED   EQU   *
         XR    R2,R2                  Set rc to zero
RETURNØØ EQU   *
         STORAGE RELEASE,             release workarea storage         X
               LENGTH=WORKLEN,        this much                        X
               ADDR=(R13),            address in R13                   X
               SP=Ø,KEY=8,            subpool Ø storage key 8          X
               COND=NO                unconditional
         LR    R15,R2                 set rc
         PR                           return
*
*
*-------------------------------------------------------------------*
* CONSTANTS VARIABLES AND DSECTS
*-------------------------------------------------------------------*
*
ASCEND   CLC   Ø(R9,R6),Ø(R11)        Executed compare
DESCEND  CLC   Ø(R9,R11),Ø(R6)        Executed compare
MOVETEMP MVC   TEMPREC(Ø),Ø(R7)       Executed move
MOVEBACK MVC   Ø(Ø,R7),TEMPREC        Executed move
MOVESWAP MVC   Ø(Ø,R7),Ø(R8)          Executed move
MOVER6   MVC   R6REC(Ø),Ø(R6)         Executed move
MOVER11  MVC   R11REC(Ø),Ø(R11)       Executed move
*
TABLE    DS    ØCL256
         DC    X'ØØØ1Ø2Ø3Ø4Ø5Ø6Ø7Ø8Ø9ØA0BØCØDØEØF'
         DC    X'1Ø111213141516171819 1A1B1C1D1E1F'
         DC    X'2Ø21222324252627282 92A2B2C2D2E2F'
         DC    X'3Ø313233343536373839 3A3B3C3D3E3F'
         DC    X'4Ø4142434445464748494A4B4C4D4E4F'
         DC    X'5Ø5152535455565758595A5B5C5D5E5F'
         DC    X'6Ø6162636465666768696A6B6C6D6E6F'
         DC    X'7Ø7172737475767778797A7B7C7D7E7F'
         DC    X'8Ø8182838485868788898A8B8C8D8E8F'
         DC    X'9Ø9192939495969798999A9B9C9D9E9F'
         DC    X'AØA1A2A3A4A5A6A7A8A9AAABACADAEAF'
         DC    X'BØB1B2B3B4B5B6B7B8B9BABBBCBDBEBF'
         DC    X'CØFAFBFCFDFEFFC7C8C9CACBCCCDCECF'
         DC    X'DØD1D2D3D4D5D6D7D8D9DADBDCDDDEDF'
         DC    X'EØE1E2E3E4E5E6E7E8E9EAEBECEDEEEF'
         DC    X'FØF1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'
```

```
*
WORKAREA DSECT
SAVEAREA DS    18D                    save area
NUMRECS  DS    F                      Number of records
RECLEN   DS    F                      Record length
FLDOFF   DS    F                      Field offset
FLDLEN   DS    F                      Field length
HIGHINT  DS    F                      High Interval Number
INTINDX  DS    F                      Number records / 9
IINDEX   DS    F                      Current index
JINDEX   DS    F                      Adjust index
KINDEX   DS    F                      temp index
SORTTYPE DS    X
ISHEX    EQU   X'8Ø'                  Is hex
TEMPREC  DS    CL255                  Temporary record store
R6REC    DS    CL255                  Temporary Record (R6)
R11REC   DS    CL255                  Temporary Record (R11)
WORKLEN  EQU   *-WORKAREA
*
         YREGS
*
         END
```

Rob Scott
MVS Consultant (UK)

Using VSAM information for reorganizations

There are several reasons why a VSAM dataset might need to be reorganized, including the following:

- Too many extents
- CA splits
- Too many CI splits
- The dataset is residing on the wrong disk.

Some people know everything about their VSAM datasets, but I'm not one of them. However, there are some things about a dataset that are useful to know, for example, information about its usage, such as how many records are inserted/deleted or

updated, and how many records there are in the dataset. If a lot of records are inserted or deleted in one day, the dataset will need to be reorganized.

The space allocation of the dataset is also important. For example, if you have a dataset with 15 extents of 500 cylinders you won't need to reorganize, whereas if the dataset has extents of four cylinders, reorganizing will improve performance.

Too many CI and CA splits spell disaster for the dataset. And too many extents consume storage in CICS, so you can improve performance by minimizing the extents.

If two heavy I/O-consuming datasets reside on one disk, you'll want to spread the I/O in order to get better throughput.

In our company, some datasets are reorganized every day, others once a week, and the rest once a month. If any dataset needs to be reorganized more frequently, it will move to the job that runs more frequently (and conversely if it needs to be reorganized less frequently).

Once a week, the job presented in this article gives me the information I need. The coding is written in PL/I and will be executed by JCL. If you're using another level of IDCAMS you'll need to change the record-layout of LC1, LC2, LC3, and LC4 to meet your requirements. THE PL/I PROGRAM

```
/****** VSAM INFO GIVING ADVICE FOR REORGANISATION    =VSMINFO= ******/
/*                                                                    */
/*  PROGRAM        : VSMINFO                                          */
/*                                                                    */
/*  PURPOSE        : GET USEFUL INFORMATION NEEDED TO KNOW IF YOU     */
/*                   NEED TO REORGANISE VSAM DATASETS                 */
/*                                                                    */
/*  INPUT          : IDCAMS : LISTOUTPUT FROM IDCAMS                  */
/*                                                                    */
/*  OUTPUT         : PRINTØ1 : LIST YOU WANT TO HAVE                  */
/*                                                                    */
/*  LAST CHANGE / AUTHOR : PST  02-Ø7-3Ø  CREATION                    */
/*                                                                    */
/*  NOTES          : NONE                                             */
/*                                                                    */
/****** VSAM INFO GIVING ADVICE FOR REORGANISATION    =VSMINFO= ******/
```

```
1VSMINFO:PROC OPTIONS(MAIN) REORDER;
 DCL IDCAMS FILE RECORD  SEQL INPUT ;
 DCL PRINTØ1 FILE RECORD  SEQL OUTPUT;
 ON ENDFILE (IDCAMS)     EOF      = '1'B;

 /* OUTPUTRECORD FROM IDCAMS                    */
 DCL REC                 CHAR(131) INIT (' ');

 /* FIND TYPE: INDEX /DATA AND DATASETNAME     */
 DCL 1 LC1               BASED(ADDR(REC)),
       2 DATA_INDEX      CHAR(Ø5),
       2 NVT1            CHAR(Ø2),
       2 FIND_NAME       CHAR(Ø9),
       2 NVT2            CHAR(Ø1),
       2 NAME            CHAR(44);

 /* USED TO DETECT RECORDS SPLITS AND EXTENTS  */
 DCL 1 LC2               BASED(ADDR(REC)),
       2 NVT1            CHAR(12),
       2 FIND_RECS       CHAR(Ø8),
       2 NVT2            CHAR(Ø2),
       2 RECS            CHAR(1Ø),
       2 NVT3            CHAR(Ø5),
       2 FIND_SPLITS     CHAR(Ø9),
       2 NVT4            CHAR(12),
       2 SPLITS          CHAR(Ø3),
       2 NVT5            CHAR(27),
       2 EXTENTS         CHAR(Ø2);

 /* FIND SPACE PARAMETERS XX CYLS/TRKS         */
 DCL 1 LC3               BASED(ADDR(REC)),
       2 NVT1            CHAR(Ø8),
       2 FIND_SPACE      CHAR(1Ø),
       2 NVT2            CHAR(Ø6),
       2 SPACE_TYPE      CHAR(Ø3),
       2 NVT3            CHAR(Ø1),
       2 SPACE           CHAR(Ø4);

 /* FIND LRECL                                  */
 DCL 1 LC4               BASED(ADDR(REC)),
       2 NVT1            CHAR(37),
       2 FIND_LRECL      CHAR(Ø8),
       2 NVT2            CHAR(11),
       2 LRECL           CHAR(Ø5);

 /* OUTPUT LINES: HEADERS                       */
 DCL 1 K1,
       2 ASA             CHAR(Ø1) INIT('1'),
       2 TEXT1           CHAR(1Ø) INIT(' * * * *  '),
       2 TEXT2           CHAR(29) INIT(' V S A M - I N F O R M A T I'),
```

```
      2 TEXT3           CHAR(28) INIT('O N   R E O R G A N I Z E'),
      2 TEXT4           CHAR(19) INIT('    D A T A S E T S'),
      2 TEXT5           CHAR(10) INIT('  * * * * '),
      2 TEXT6           CHAR(09) INIT(' '),
      2 TEXT7           CHAR(07) INIT(' DATE:'),
      2 DATUM1          CHAR(08) INIT(' '),
      2 TEXT8           CHAR(04) INIT(' '),
      2 TEXT9           CHAR(05) INIT('PAGE '),
      2 PAGE            PIC'ZZ9' INIT(0);

DCL 1 K2,
      2 ASA             CHAR(01) INIT('-'),
      2 TEXT1           CHAR(44) INIT('DATASET-NAME'),
      2 NVT1            CHAR(01) INIT(' '),
      2 TEXT2           CHAR(07) INIT('CI - CA'),
      2 NVT2            CHAR(01) INIT(' '),
      2 TEXT3           CHAR(04) INIT('EXT.'),
      2 NVT3            CHAR(02) INIT(' '),
      2 TEXT4           CHAR(22) INIT('------------------ REC'),
      2 TEXT5           CHAR(21) INIT('ORDS ---------------'),
      2 NVT4            CHAR(02) INIT(' '),
      2 TEXT6           CHAR(14) INIT('---- SPACE --'),
      2 NVT5            CHAR(02) INIT(' '),
      2 TEXT7           CHAR(11) INIT('-- LRECL --');

DCL 1 K3,
      2 ASA             CHAR(01) INIT(' '),
      2 NVT1            CHAR(45) INIT(' '),
      2 TEXT2           CHAR(07) INIT(' SPLITS'),
      2 NVT2            CHAR(07) INIT(' '),
      2 TEXT4           CHAR(22) INIT('     TOTAL   INSERTED '),
      2 TEXT5           CHAR(21) INIT('   DELETED    UPDATED'),
      2 NVT4            CHAR(16) INIT(' '),
      2 TEXT7           CHAR(13) INIT('  AVG.   MAX.');

/* OUTPUT LINES : DETAIL LINES                          */
DCL 1 D1,
      2 ASA             CHAR(01),
      2 NAME            CHAR(44),
      2 NVT1            CHAR(01),
      2 CI_SPLITS       CHAR(03),
      2 NVT2            CHAR(01),
      2 CA_SPLITS       CHAR(03),
      2 NVT3            CHAR(02),
      2 EXTENTS         CHAR(02),
      2 NVT4            CHAR(03),
      2 TOT_RECS        CHAR(10),
      2 NVT5            CHAR(01),
      2 INS_RECS        CHAR(10),
      2 NVT6            CHAR(01),
```

```
      2 DEL_RECS         CHAR(10),
      2 NVT7             CHAR(01),
      2 UPD_RECS         CHAR(10),
      2 NVT8             CHAR(02),
      2 SPACE_TYPE       CHAR(03),
      2 TEXT1            CHAR(01),
      2 SPACE_PRI        CHAR(04),
      2 TEXT2            CHAR(01),
      2 SPACE_SEC        CHAR(04),
      2 TEXT3            CHAR(01),
      2 NVT9             CHAR(01),
      2 AV_RECSIZE       CHAR(05),
      2 NVT10            CHAR(02),
      2 MAX_RECSIZE      CHAR(05);
DCL CATNAME              CHAR(08) BASED(ADDR(D1.NAME));
/* INITIALIZE DETAIL LINE                  */
D1 = ' ';
D1.TEXT1  = '(';
D1.TEXT2  = ',';
D1.TEXT3  = ')';
DCL (ADDR,DATE,SUBSTR)   BUILTIN;
/* HELP FIELDS                              */
DCL EOF                  BIT (01) INIT ('0'B);
DCL I                    FIXED BIN (15);
DCL RECORD_COUNT         FIXED BIN (15) INIT(99);
/* START MAIN PROGRAM                       */
K1.DATUM1 = DATE;
K1.DATUM1 = SUBSTR(K1.DATUM1,5,2) || '-' ||
            SUBSTR(K1.DATUM1,3,2) || '-' ||
            SUBSTR(K1.DATUM1,1,2);
OPEN FILE (IDCAMS),
     FILE (PRINT01);
READ FILE (IDCAMS) INTO (REC);
DO   WHILE (¬EOF);
     /* SKIP USELESS LINES                  */
     DO   WHILE (LC1.FIND_NAME ¬= '--------' &
                 ¬EOF);
          REC = ' ';
          READ FILE (IDCAMS) INTO (REC);
          END;
     IF ¬EOF
     THEN DO;
          /* WE HAVE LINES WE COULD USE     */
          D1.NAME = LC1.NAME;
          IF CATNAME = 'CATALOG.'
          THEN DO;
               RECORD_COUNT = 99;
               REC = ' ';
               READ FILE (IDCAMS) INTO (REC);
               END;
```

```
ELSE DO;
     /* RETRIEVE AND FILL THE DATA */
     DO   WHILE (LC4.FIND_LRECL ¬= 'AVGLRECL' &
                 ¬EOF);
          REC = ' ';
          READ FILE (IDCAMS) INTO (REC);
          END;
     DO   I = 1 TO 5 WHILE (SUBSTR(LC4.LRECL,I,1) = '-');
          SUBSTR(LC4.LRECL,I,1) = ' ';
          END;
     D1.AV_RECSIZE = LC4.LRECL;
     IF D1.AV_RECSIZE = '    Ø'
     THEN DO;
          D1.AV_RECSIZE  = ' ';
          D1.MAX_RECSIZE = ' ';
          END;
     ELSE DO;
          DO   WHILE (LC4.FIND_LRECL ¬= 'MAXLRECL' &
                      ¬EOF);
               REC = ' ';
               READ FILE (IDCAMS) INTO (REC);
               END;
          DO   I = 1 TO 5
                     WHILE (SUBSTR(LC4.LRECL,I,1) = '-');
               SUBSTR(LC4.LRECL,I,1) = ' ';
               END;
          D1.MAX_RECSIZE = LC4.LRECL;
          END;
     DO   WHILE (LC2.FIND_SPLITS ¬= 'SPLITS-CI' &
                 ¬EOF);
          REC = ' ';
          READ FILE (IDCAMS) INTO (REC);
          END;
     DO   I = 1 TO 3 WHILE (SUBSTR(LC2.SPLITS,I,1) = '-');
                SUBSTR(LC2.SPLITS,I,1) = ' ';
                END;
     D1.CI_SPLITS = LC2.SPLITS;
     DO   I = 1 TO 1Ø WHILE (SUBSTR(LC2.RECS,I,1) = '-');
          SUBSTR(LC2.RECS,I,1) = ' ';
          END;
     D1.TOT_RECS = LC2.RECS;
     DO   WHILE (LC2.FIND_SPLITS ¬= 'SPLITS-CA' &
                 ¬EOF);
          REC = ' ';
          READ FILE (IDCAMS) INTO (REC);
          END;
     DO  I = 1 TO 1Ø WHILE (SUBSTR(LC2.RECS,I,1) = '-');
         SUBSTR(LC2.RECS,I,1) = ' ';
         END;
     D1.DEL_RECS = LC2.RECS;
```

```
DO  I = 1 TO 3 WHILE (SUBSTR(LC2.SPLITS,I,1) = '-');
    SUBSTR(LC2.SPLITS,I,1) = ' ';
    END;
D1.CA_SPLITS = LC2.SPLITS;
DO  I = 1 TO 2 WHILE (SUBSTR(LC2.EXTENTS,I,1) = '-');
    SUBSTR(LC2.EXTENTS,I,1) = ' ';
    END;
D1.EXTENTS = LC2.EXTENTS;
DO   WHILE (LC2.FIND_RECS ¬= 'INSERTED' &
           ¬EOF);
     REC = ' ';
     READ FILE (IDCAMS) INTO (REC);
     END;
DO  I = 1 TO 10 WHILE (SUBSTR(LC2.RECS,I,1) = '-');
    SUBSTR(LC2.RECS,I,1) = ' ';
    END;
D1.INS_RECS = LC2.RECS;
DO   WHILE (LC2.FIND_RECS ¬= 'UPDATED-' &
           ¬EOF);
     REC = ' ';
     READ FILE (IDCAMS) INTO (REC);
     END;
DO  I = 1 TO 10 WHILE (SUBSTR(LC2.RECS,I,1) = '-');
    SUBSTR(LC2.RECS,I,1) = ' ';
    END;
D1.UPD_RECS = LC2.RECS;
DO   WHILE (LC3.FIND_SPACE ¬= 'SPACE-TYPE' &
           ¬EOF);
     REC = ' ';
     READ FILE (IDCAMS) INTO (REC);
     END;
IF LC3.SPACE_TYPE = 'CYL'
THEN D1.SPACE_TYPE = 'CYL';
ELSE D1.SPACE_TYPE = 'TRK';
DO   WHILE (LC3.FIND_SPACE ¬= 'SPACE-PRI-' &
           ¬EOF);
     REC = ' ';
     READ FILE (IDCAMS) INTO (REC);
     END;
DO  I = 1 TO 4 WHILE (SUBSTR(LC3.SPACE,I,1) = '-');
    SUBSTR(LC3.SPACE,I,1) = ' ';
    END;
D1.SPACE_PRI = LC3.SPACE;
DO   WHILE (LC3.FIND_SPACE ¬= 'SPACE-SEC-' &
           ¬EOF);
     REC = ' ';
     READ FILE (IDCAMS) INTO (REC);
     END;
DO  I = 1 TO 4 WHILE (SUBSTR(LC3.SPACE,I,1) = '-');
    SUBSTR(LC3.SPACE,I,1) = ' ';
```

```
                END;
              D1.SPACE_SEC = LC3.SPACE;
              /* PRINT ROUTINE                        */
              IF RECORD_COUNT > 6Ø
              THEN DO;
                   RECORD_COUNT  = 4;
                   K1.PAGE = K1.PAGE + 1;
                   WRITE FILE(PRINTØ1) FROM(K1);
                   WRITE FILE(PRINTØ1) FROM(K2);
                   WRITE FILE(PRINTØ1) FROM(K3);
                   D1.ASA = 'Ø';
                   END;
              WRITE FILE (PRINTØ1) FROM (D1);
              D1.ASA = ' ';
              RECORD_COUNT = RECORD_COUNT + 1;
              END;
         END;
     END;
 CLOSE FILE (IDCAMS),
       FILE (PRINTØ1);
 END VSMINFO;
```

THE JCL

```
//STREAM1   JOB CRC-57ØØ-Ø,'VSAM INFO',
//             CLASS=A,MSGCLASS=A
//*
//* PROJECT    : DISK UTILITIES
//* JOB        : LIST DATA TO KNOW WHICH vsam DATASET HAS TO BE
//*              REORGANISED
//*
//* STEPS      : NR |PROGRAM  |FUNCTION
//*              --|--------|-------
//*              1Ø |IDCAMS   |LIST CATALOG(S)
//*              2Ø |VSMINFO  |PRINT VSAM INFO AND STATISTICS
//*
//* SYSOUTSRT. : 1 : ON PAPER
//* NOTES      : NONE
//* LIST CATALOGS IN WHICH THE VSAM DATASET ENTRIES IN EXIST
//STEP1Ø       EXEC PGM=IDCAMS
//SYSPRINT  DD DSN=IDCAMS.OUTPUT,DISP=(,CATLG),
//             DCB=(LRECL=131,BLKSIZE=279Ø7,RECFM=VBA),
//             UNIT=SYSDA,SPACE=(CYL,(1Ø,1Ø))
//SYSIN     DD *
    LISTCAT CATALOG (CATALOG.VSMICF1.VDISK1) -
        DATA INDEX ALL ;
    LISTCAT CATALOG (CATALOG.VSMICF1.VDISK2) -
        DATA INDEX ALL ;
/*
```

```
//*
//* REFORMAT THE OUTPUT OF IDCAMS TO GET THE DATA WE WANT TO SEE
//* IF A DATASET HAS TO BE REORGANISED
//STEP2Ø  EXEC PGM=VSMINFO
//IDCAMS    DD DSN=IDCAMS.OUTPUT,DISP=(OLD,DELETE)
//PRINTØ1   DD SYSOUT=1,DCB=(LRECL=137,RECFM=VA)
//SYSPRINT  DD SYSOUT=*
//*
```

Teun Post
(The Netherlands)

A make utility to generate JCL

MULTI-COMPONENT MODULES

In modular programming disciplines, the logic components (the source code) are usually kept separately in individual modules. A separate source is even mandatory in some programming languages like COBOL for functions/subroutines. The most frequently used standardized program segments or the data structures (include files/copy books) are kept in different source modules. Some load module generation creates intermediate temporary files as output from the compiler, which is input to a Linkage Editor to produce the final load module. But there are other generation styles, which create object code from source code into an object library, and then one final Linkage Editor phase to create a load module. Dealing with multi-components becomes a necessity in object-oriented development like the C++ language.

COMPILING PROBLEMS

Dealing with multi-components creates manageability and consistency problems. If an include files/copy books changes, then all source code using this must be reprocessed (be

recompiled). If you change a subroutine/function, all NODYNAM load modules using it must be re-link edited. Therefore you have to manage your JCL to respond to this request. In practice 'builder' JCL is kept in a library and, on request, either it will re-edit the JCL to comment out unaffected files or it will be submitted as is, causing some extra modules to be processed. There are products commercially available to track and oversee this kind of problem, but, if you don't have them, the situation can become very tedious.

SOLUTION

If we look around we will find the **make** utility, which is familiar to programmers who are involved with C/C++ programming on Unix/PC-DOS platforms. To my knowledge, there is no equivalent tool in MVS (unless you're using the Unix shell). Therefore I decided to develop a **make** utility that will run on TSO/ISPF to produce the necessary JCL to control multi-module projects. My primary objective is to prepare an environment for C/C++ development, but the **make** utility may be used in the compile process of other programming languages.

DESIGN CRITERIA

The first problem is to find a way to obtain the last change (modification) time information for PDS/PDSE members. There are no system utilities (to my knowledge) that can do this. VTOC information is not suitable for this purpose. Therefore I decided to use ISPF member statistics to gather the last-change time stamp. The object modules are also kept in a PDS/PDSE. Since they are generated by a compiler, the information is not available and will be deleted after every regeneration. Fortunately, the C/C++ compiler puts time stamps in a TXT line of the object deck. (I did not check out the other compilers' object decks to see if it is valid for them.) Therefore I can read the object deck and localize the TXT line to gather the time stamp. With the help of ISPF LIB services, I can get statistics of PDS members with a time stamp of the object deck. This will simplify things by

scanning the dependency time stamp. The source module may be set as STATISTICS ON either in its ISPF edit phase (I do not think any TSO edit is used at all for source update) or it will be set during the first scan time with ISPF LIB services. I could not find any solution to obtain a time stamp for LKED modules. However, I made an assumption that for any object module generated within a dependency set, there will be no need to generate a load module. This will be handled by means of a proper arrangement of macro variables. The load module (DCB=(RECFM=U(B),ORG=PO..) type PDS) members will be generated without checking the time stamp against the dependency time stamp.

The implemented solution must behave like standard **make** utilities. But **make** utility standards depend on hierarchical filesystem formats and we are dealing with PDS/PDSE members. Parentheses are most often used instead of the IBM conventional format, which is used in header file definitions. This makes a small difference in the syntax of component representation. Members must be surrounded by parentheses if they are accompanied by a file name (library).

In the **make** utility, the following concepts were implemented:

- Provide and use macro variable definition, substitution, re-assignment, and recursive (assignment made with :=).
- Allow nested inclusion of commands via **|include x** directive, where x is the path name or member name (default to current library of make object).
- Provide conditional directives like |if, |else, |endif.
- Provide condition testing =,¬=,>,<,>=, and <= for |if.
- Provide rules definitions (rule lines may be controlled with |if, |else, |endif).
- Provide dependency definitions.
- Provide comment line.
- Provide line continuation (terminating with \ ).

- Provide capability to differentiate member, library, and path of component.
- Provide capability to generate repeated lines for each dependent components starting with the second one.
- Provide references to both target and dependent components.

The following example may help to improve your understanding of how **make** will help us to create JCL easily for multi-component projects.

Assuming that the components are distributed as:

- Project source codes MYPROG, SRC1, SRC2, SRC3 in hlq.mylib.src.c (dependency components).
- Project header files HDR1, HDR2, HDR3, HDR4 in hlq.mylib.src.h (dependency components).
- Project object modules MYOBJ, SRC1OBJ1, SRC2OBJ2, SRC3OBJ3 in hlq.mylib.obj (target components).
- Project load modules MYPROGLD in hlq.mylib.loadlib.
- MYPROG is using HDR1, HDR2, HDR4.
- SRC1 is using HDR1, HDR3.
- SRC2 is using HDR2, HDR4.
- SRC3 is using HDR1, HDR2, HDR3, HDR4.
- Object MYOBJ is output from procedure CMPLPROC compiling MYPROG.
- Object SRC1OBJ1 is output from procedure CMPLPROC compiling SRC1
- Object SRC2OBJ2 is output from procedure CMPLPROC compiling SRC2.
- Object SRC3OBJ3 is output from procedure CMPLPROC compiling SRC3.
- Load module MYPROGLD produced from the LKED phase

by binding MYOBJ, SRC1OBJ1, SRC2OBJ2, SRC3OBJ3 objects and named as MYPROGLD.

- A stand-alone COBOL program COBPROG (no need to create object deck).

Assuming that we have similar library structures in development, test, and production environments, and we want to regenerate in each environment, we keep the following text code in library sys.make:

```
---- "sys.make(envdef)"
* define environment for development
env := D
------eof----
----"sys.make(libdef)"
* get environment
|include envdef
* construct high-level qualifiers for environment
|if ($env) = P
prj  :=  prodhlq
|endif
|if ($env) = T
prj  :=  testhlq
|else
prj  :=  hlq
|endif
* define libraries shortcuts
lib  := ($prj).mylib
src := ($lib).src.c
hdr := ($lib).src.h
obj := ($lib).obj
Lkd := ($lib).loadlib
----eof------
--------"sys.make(rules)"
*get library short cuts
|include libdef
*define rule for object target constructed from "C" and "H" sources
.o.c  CMPLPROC, \
//   INPUT='($.C)', \
//   OUTFILE='($.O)',\
//   CPARM='NOSOURCE NOMARGIN NOSEQ LOC'
*define how we link edit the object components
.go.o LKEDPROC, \
//   INPUT=($.O), \
//   OUTPUT=($L.GO) \
//MYLIB  DD DISP=SHR,DSN=($obj) \
//SYSIN DD *. \
   INCLUDE MYLIB($M.O) \
```

```
   INCLUDE MYLIB($*M.O) \
   NAME ($M.GO)(R) \
/*
*define rules for COBOL  II
.go.cob2 COBPROC
//  INPUT='($.cob2)',
// OUTPUT=($.go)
----eof--------
Now define a make file in our source library (to prevent extra
navigation to other library)
---- "hlq.mylib.src(mymake)" ----
* get standard rules
|include sys.make(rules)
*now define dependencies SRC1,HDR1,HDR3 of target SRC1OBJ
($obj)( SRC1OBJ1).O    :   ($src)(SRC1).C \
                                          ($hdr)(HDR1).H \
                                          ($hdr)(HDR3).H
* define SRC2OBJ2 dependencies
($obj)( SRC2OBJ2).O    :   ($src)(SRC2).C \
                                          ($hdr)(HDR2).H \
                                          ($hdr)(HDR4).H
* define SRC3OBJ3 dependencies
 ($obj)( SRC3OBJ3).O    :   ($src)(SRC3).C \
                                          ($hdr)(HDR1).H \
                                          ($hdr)(HDR2).H \
                                          ($hdr)(HDR3).H \
                                          ($hdr)(HDR4).H
* define MYOBJ dependencies
 ($obj)(MYOBJ).O          :   ($src)(MYPROG).C \
                                          ($hdr)(HDR1).H \
                                          ($hdr)(HDR2).H \
                                          ($hdr)(HDR4).H
* Now combine all this object to construct LOAD module
($lkd)(MYPROGLD).GO  :  ($obj)(MYOBJ).O \
                        ($obj)( SRC1OBJ1).O \
                        ($obj)( SRC2OBJ2).O \
                        ($obj)( SRC3OBJ3).O
*Now create load module from COBOL
($lkd)(COBMAIN).GO     :  ($src)(COBPROG)
----eof ------
```

Now we can create the required JCL by using a simple command:

```
TSO MAKE hlq.mylib.src(mymake)
```

Or we can simply write the line command **make** on member **mymake** in the ISPF member list of hlq.mylib.src.

Assume we changed only the HDR3 header file and wish to produce the necessary JCL to create MYPROGLD after all related compilations. The **make** command for mymake will produce and submit the following JCL:

```
//MYJOBA JOB 'MAKE',MSGCLASS=X,NOTIFY=MYUID,TYPERUN=HOLD
//STEP1 EXEC CMPLPROC,
//    INPUT='HLQ.MYLIB.SRC.C(SRC1)',
//    OUTFILE='HLQ.MYLIB.OBJ(SRC1OBJ1)',
//    CPARM='NOSOURCE NOMARGIN NOSEQ LOC'
//STEP2 EXEC CMPLPROC,
//    INPUT='HLQ.MYLIB.SRC.C(SRC3)',
//    OUTFILE='HLQ.MYLIB.OBJ(SRC3OBJ3)',
//    CPARM='NOSOURCE NOMARGIN NOSEQ LOC'
//STEP3 EXEC LKEDPROC,
//    INPUT=HLQ.MYLIB.OBJ(MYOBJ),
//    OUTPUT=HLQ.MYLIB.LOADLIB
//MYLIB  DD DISP=SHR,DSN=HLQ.MYLIB.OBJ
//SYSIN DD *
   INCLUDE MYLIB(MYOBJ)
   INCLUDE MYLIB(SRC1OBJ1)
   INCLUDE MYLIB(SRC2OBJ2)
   INCLUDE MYLIB(SRC3OBJ3)
   NAME  MYPROGLD(R)
/*
```

If we changed only the SRC1 module then **make** will produce:

```
//MYJOBA JOB 'MAKE',MSGCLASS=X,NOTIFY=MYUID,TYPERUN=HOLD
//STEP1 EXEC CMPLPROC,
//    INPUT='HLQ.MYLIB.SRC.C(SRC1)',
//    OUTFILE='HLQ.MYLIB.OBJ(SRC1OBJ1)',
//    CPARM='NOSOURCE NOMARGIN NOSEQ LOC'
//STEP2 EXEC LKEDPROC,
//    INPUT=HLQ.MYLIB.OBJ(MYOBJ),
//    OUTPUT=HLQ.MYLIB.LOADLIB
//MYLIB  DD DISP=SHR,DSN=HLQ.MYLIB.OBJ
//SYSIN DD *
   INCLUDE MYLIB(MYOBJ)
   INCLUDE MYLIB(SRC1OBJ1)
   INCLUDE MYLIB(SRC2OBJ2)
   INCLUDE MYLIB(SRC3OBJ3)
   NAME  MYPROGLD(R)
/*
```

I think this is the simplest and cheapest way to cope with multi-component projects. It will even make life easier with commercial version control products like CCC. In our case just edit your mymake dependency file and issue a **make** command (which will understand the edit environment and behave as an edit macro to acquire dataset and member name to generate and submit the requested JCL). If your site policy requires you to use products like CCC to promote assets from development to test

and production environment, you simply transfer source code without any procedure associated with them and transfer your dependency file with the associated proc makeproc, which is a stored procedure that invokes REXX under batch environment producing the necessary JCL set to achieve the desired result.

REXX IMPLEMENTATION OF MAKE

Here is the **make** utility for MVS, implemented in REXX.

The current version will understand commands in the following formats:

```
Make library(member) action
Make library member action
```

Where the action could be:

- SUBMIT – submit created JCL.
- LIST – lists only created JCL.
- MACRO – list defined macro variables and values.
- RULES – lists defined rules.
- DEPENDENCY – list given dependencies.

MAKE REXX

```
/* REXX */
 PARSE UPPER ARG DSN MEMBER SERV
IF DSN='' THEN DO
  ADDRESS ISPEXEC  CONTROL  ERRORS  RETURN
  ADDRESS ISREDIT RESET
  IF RC>2Ø THEN DO
   INEDIT=1
   Y=SETLANG(ENU)
   ADDRESS ISREDIT
   'MACRO PROCESS(PRM)'
   '(MEMBER) = MEMBER'
   '(DSN) = DATASET'
   IF Y<>'CHT' THEN DO
     'F NEXT " "'
     'LABEL .ZCSR = .CMPL ØØØ1'
   END
```

```
  END
ELSE DO
  SAY 'Usage: MAKE Dataset(Member) SUBMIT/LIST/MACRO/RULES'
  RETURN 8
END
 IF INDEX(DSN,'(')>Ø THEN DO
    SERV=MEMBER
    PARSE UPPER VAR DSN DSN '(' MEMBER ')'
 END
 LIBID.=''   /* ISPF LIB HDLR */
 LIBID.Ø=Ø
 DSN=STRIP(DSN,B,"'")
 DROPBUF Ø
 MAKEBUF
 LNCNT=Ø
 UID=SYSVAR(SYSUID)
 IF SERV='' THEN SERV=''
 QUEUE "//"UID"A JOB ,'MAKE',MSGLEVEL=(1,1),MSGCLASS=X,REGION=5M"
 /* "ORDER=(DBSA.CPP.PROCLIB,SYS2.NPROCLIB,SYS2.PROCLIB)" */
 QUEUE "//LIBS JCLLIB "||,
  "ORDER=(CBC.SCBCPRC,SYS2.NPROCLIB,SYS2.PROCLIB)"
 STP_CNT=Ø
 FID.Ø=Ø
 CND.Ø=Ø
 VS.Ø=Ø
 RL.Ø=Ø
ADDRESS ISPEXEC  CONTROL  ERRORS  RETURN
 VN.=''                 /* MACRO VARIABLES                   */
 VS.=''                 /* MACRO VARIABLES SUBSTITION VALUES*/
 RN.=''                 /* RULES                             */
 DN.=''                 /* DEPENDENCIES                      */
 TN.=''                 /* DEPENDENC TOKENS */
 DN.Ø=Ø;TN.Ø=Ø;RN.Ø=Ø;VN.Ø=Ø
 DHDL=PUSH_FILE_HDL(DSN,MEMBER)
 LN=GET_NEXT_REC()
 DO WHILE LN>''
   SELECT
   WHEN INDEX(LN,':=')>Ø THEN DO   /* MACRO ASSIGNMENT */
        CALL BUILD_VAR
        END
   WHEN LEFT(LN,1)='.'   THEN DO   /* RULE DEFINITION  */
        CALL BUILD_RULES
        END
   WHEN LEFT(LN,1) = '|' THEN DO   /* CONTROL MARK     */
        CALL CONTROL_DIRECTIVE
        END
   WHEN INDEX(LN,' : ') >Ø THEN DO  /* DEPENDENCY DEFIN */
        CALL BUILD_DEPENDENT
        END
```

```
   OTHERWISE
   END /* SELECT */
   LN=GET_NEXT_REC()
 END   /* DO WHILE LN>''*/
 ADDRESS ISPEXEC
 DO I=1 TO LIBID.Ø  /* CLOSE AND RELEASE HANDLERS FOR ISPF */
    DSN=LIBID.I
    DSK=LIBID.DSN.1
    "LMCLOSE DATAID("DSK")"
    "LMFREE  DATAID("DSK")"
 END
 ADDRESS TSO
 IF SERV = 'SUBMIT' THEN DO
    QUEUE "ÖÖ"
    PUSH "SUBMIT * END(ÖÖ)"
 END
 IF SERV='LIST' THEN DO
   DO WHILE QUEUED() > Ø
     PULL TXT
     SAY TXT
   END
 END
 IF SERV='MACRO' THEN DO
    DO I=1 TO VN.Ø
       NMS=VN.I
       SAY NMS ':=' VS.NMS
    END
 END
 IF SERV='RULES' THEN CALL DISPLAY_RULES
 RETURN Ø
DISPLAY_RULES:
    DO I=1 TO RN.Ø
       NMS=RN.I
       SAY 'RULE:' NMS
       DO J=1 TO RN.NMS.Ø
       SAY  RN.NMS.J
       END
    END
BUILD_VAR:      /* CREATES VARIABLE NAMES & VALUES */
 PARSE UPPER VAR LN VP OP REST
 VP=STRIP(VP,'B')
 IF OP = ':=' THEN        /* ASSIGNMENT TO VARIABLE */
    DO
    IF VS.VP='VS.VP' | VS.VP='' THEN     /* IF FIRST TIME */
     DO
       WI=VN.Ø+1
       VN.Ø=WI
       VN.WI=VP             /* KEEP MACVAR NAME       */
     END
```

```
     REST=MAC_SUBSITUDE(STRIP(REST,'B'))  /* REPLACE OTHER MAC VARS */
     VS.VP=REST           /* DEFINE MACRO VARIABLE  */
    END
   RETURN
BUILD_RULES:      /* CREATES RULES DEFINITIONS */
   LN=STRIP(LN,'T',' ')
   PARSE UPPER VAR LN RULE PROC REST
   RULE=STRIP(RULE,'B')
   PROC=STRIP(PROC,'B')
   REST=STRIP(REST,'B')
   IF RN.RULE.Ø='RN.RULE.Ø' | RN.RULE.Ø='' THEN /* NEW DEFINITION */
      DO
        WCNT=RN.Ø + 1
        RN.Ø=WCNT
        RN.WCNT=RULE
      END
   RN.RULE.=''
   RN.RULE.Ø=1
   RN.RULE.1=PROC
   DO WHILE REST = '\'
      LN=GET_NEXT_REC()
      IF LEFT(LN,1)='|' THEN
         DO
          CALL CONTROL_DIRECTIVE
          REST='\'
          ITERATE
         END
         IF LEFT(REST,1)='*' THEN
           DO
             REST='\'
             ITERATE
           END
      REST=RIGHT(LN,1)
      IF REST = '\' THEN LN=STRIP(LN,'T','\')
      WCNT=RN.RULE.Ø + 1
      RN.RULE.Ø=WCNT
      RN.RULE.WCNT=STRIP(LN,'T',' ')
   END
   RETURN
PUSH_FILE_HDL:             /* ALLOW INCLUDE (NESTED INCLUDE) */
   ARG WDSN,WMEM
   WPTH=STRIP(STRIP(WDSN,'B'),'B',"'")"("STRIP(WMEM,'B')")"
   IF SYSDSN("'"WPTH"'") ¬= 'OK' THEN RETURN ''
   X=OUTTRAP('ALLOC.')
   I=FID.Ø+1
   FID.Ø=I
   DSK='DD'I
   ADDRESS TSO "FREE FI("DSK")"
   ALLOC.Ø=Ø
```

```
   ADDRESS TSO "ALLOC FI("DSK") DSN('"WPTH"') SHR"
   X=OUTTRAP('OFF')
   IF ALLOC.Ø > Ø THEN RETURN ''
   ALLOC.=''
   RETURN DSK
POP_FILE_HDL:              /* TERMINATION OF NESTED INCLUDE  */
   IF FID.Ø = Ø THEN RETURN ''
   X=OUTTRAP('ALLOC.')
   I=FID.Ø
   DSK='DD'I
   I=I-1
   FID.Ø=I
   ADDRESS TSO "FREE FI("DSK")"
   X=OUTTRAP('OFF')
   IF ALLOC.Ø > Ø THEN RETURN ''
   DSK='DD'I
   RETURN DSK
GET_FILE_HDL:              /* CURRENT FILE HNDL              */
   I=FID.Ø
   RETURN 'DD'I
GET_NEXT_REC:              /* OBTAIN RECORD FROM CURRENT HDL */
   DSK=GET_FILE_HDL()      /* GET CURRENT FILE HDLR          */
   IF DSK='DDØ' THEN RETURN ''  /* ANY MORE FILE HDLR MEANS EOF */
   'EXECIO 1 DISKR 'DSK' (STEM WLN.'
   IF RC>Ø THEN DO         /* EOF OF THIS HDLR               */
     'EXECIO Ø DISKR 'DSK' (FINIS' /* CLOSE THIS HDLR       */
     IF POP_FILE_HDL()>'' THEN RETURN GET_NEXT_REC()
     RETURN ''
   END
 LNCNT=LNCNT+1
   RETURN STRIP(WLN.1,'T')
MAC_SUBSITUDE:            /* REPLACE ($MCVR) WITH VALUE OF MCVR */
   ARG REST
   KØ=INDEX(REST,'($',1)
   IF KØ = Ø THEN RETURN REST            /* NO ($ FOUND */
   K1=INDEX(REST,')',KØ+2)
   IF K1 = Ø THEN RETURN REST            /* NO ) FOUND */
   MACVAR=SUBSTR(REST,KØ+2,K1-KØ-2)      /* GET MACVAR NAME */
   SUBSVAL='?'                           /* UNDEFINED MCVR VALUE */
   IF VS.MACVAR¬='VS.MACVAR' THEN        /* VARIABLE NOT DEFINED */
   SUBSVAL=STRIP(VS.MACVAR,'B',"'")
   REST=LEFT(REST,KØ-1)''SUBSVAL''RIGHT(REST,LENGTH(REST)-K1)
   IF INDEX(REST,'($',1)>Ø THEN RETURN MAC_SUBSITUDE(REST)
   RETURN REST
DSN_COMPONENT:            /* PARSE FILE NAME AS LIB AND MEMBER  */
   ARG X
   DSN_CMP.Ø=Ø
   X=STRIP(X,'L',"'")
   PARSE UPPER VAR X DSN_CMP.1 '(' MEM ')'
```

```
   IF DSN_CMP.1 = '' THEN RETURN
   IF MEM='MEM' | MEM='' THEN DO
      DSN_CMP.2 = DSN_CMP.1
      DSN_CMP.1 = DSN         /* DEFAULT LIBRARY IS EXEC ARGUMENT */
      DSN_CMP.Ø = 2
   END
   ELSE DO
      DSN_CMP.2 = MEM
      DSN_CMP.Ø = 2
   END
   RETURN
TEST_CND:                 /* TEST CNTRL COMMANDS VALIDITY       */
   ARG REST
   K=INDEX(REST,'($',1)       /* ANY MACRO USAGE */
   IF K>Ø THEN REST=MAC_SUBSITUDE(REST)
   PARSE UPPER VAR REST TEST OP V1
   IF OP = 'DEFINED' & VS.V1¬='VS.V1' & VS.V1¬='' THEN RETURN 'TRUE'
   IF OP = 'UNDEFINED' & VS.V1='VS.V1' & VS.V1 ='' THEN RETURN 'TRUE'
   IF OP = 'EXIST' & SYSDSN(STRIP(V1,'B'))='OK' THEN RETURN 'TRUE'
   IF OP = '='    & TEST =  V1 THEN RETURN 'TRUE'
   IF OP = '>'    & TEST >  V1 THEN RETURN 'TRUE'
   IF OP = '<'    & TEST <  V1 THEN RETURN 'TRUE'
   IF OP = '>='   & TEST >= V1 THEN RETURN 'TRUE'
   IF OP = '<='   & TEST <= V1 THEN RETURN 'TRUE'
   IF OP = '¬='   & TEST ¬= V1 THEN RETURN 'TRUE'
   RETURN 'FALSE'
CONTROL_DIRECTIVE:
  PARSE UPPER VAR LN CMD REST
  SELECT
    WHEN CMD='|INCLUDE' THEN DO
         REST=MAC_SUBSITUDE(STRIP(REST,'B'))
         DSN_CMP.Ø=Ø
         CALL DSN_COMPONENT  REST
         IF DSN_CMP.Ø = Ø THEN RETURN
         X=PUSH_FILE_HDL(DSN_CMP.1,DSN_CMP.2)
    END                           /* WHEN CMD='|INCLUDE'      */
    WHEN CMD='|IF' THEN DO
         WI=CND.Ø+1
         CND.Ø=WI
         CND.WI=TEST_CND(REST)
         IF CND.WI = 'FALSE' THEN
            DO UNTIL LEFT(LN,4)='|END'|LN=''|LEFT(LN,4)='|ELSE'
               LN=GET_NEXT_REC()
            END
    END                           /* WHEN CMD='|IF            */
    WHEN CMD='|ELSE' THEN DO
         IF CND.WI = 'TRUE' THEN
            DO UNTIL LEFT(LN,4) = '|END' | LN='' /* SKIP BLOCK   */
               LN=GET_NEXT_REC()
```

```
          END
    END                          /* WHEN CMD='|ELSE          */
    WHEN CMD='|END' THEN DO
         WI=CND.Ø-1
         CND.Ø=WI
    END                          /* WHEN CMD='|END           */
    OTHERWISE
  END                            /*  SELECT                  */
  RETURN
SUBS_DPND:
 ARG WINS,WS,WR
 UPPER WS
 WLM=WR
 WK=INDEX(WR,WS)               /* ANY USE IN THIS RULE LINE   */
 IF WK>Ø THEN DO
   WLM=INSERT(WINS,LEFT(WR,WK-1),WK-1)
   WLM=WLM''SUBSTR(WR,WK+LENGTH(WS),LENGTH(WR)-LENGTH(WS)-1)
 END
 RETURN WLM
BUILD_DEPENDENT:
   PARSE UPPER VAR LN TARGET ':' SOURCE REST
   TARGET=STRIP(TARGET,'B',' ')
   SOURCE=STRIP(SOURCE,'B',' ')
   REST=STRIP(REST,'B',' ')
   CALL ANALYSE_TOKEN TARGET
   TRULE=TKN.1; TL=TKN.2; TM=TKN.3; TP=TKN.4; TTIME=TKN.5;
   CALL ANALYSE_TOKEN SOURCE
   DRULE=TKN.1; SL=TKN.2; SM=TKN.3; SP=TKN.4; STIME=TKN.5;
   CRE_JCL=''
   IF TTIME < STIME THEN CRE_JCL = 'YES ('SM')'
   DN.=''
   DN.Ø=Ø
   LK=REST
   DO WHILE 1                    /* SCAN FOR OTHER DEPENDENCIES */
      PARSE UPPER VAR REST LN REST
      IF LN='\' THEN DO
         REST=GET_NEXT_REC()
         IF LEFT(REST,1)='*' THEN REST='\'
         LN=REST
         IF LEFT(LN,1)='|' THEN DO
            CALL CONTROL_DIRECTIVE
            REST='\'
         END
         ITERATE
      END
      IF LN='' THEN LEAVE
      IF LEFT(LN,1)¬='#' THEN
        DO
          CALL ANALYSE_TOKEN LN
```

```
          VCNT=DN.Ø + 1
          DN.Ø=VCNT
          DN.VCNT.Ø=5
          DN.VCNT.1=STRIP(TKN.1,'T',' ')    /* RULE       */
          DN.VCNT.2=STRIP(TKN.2,'T',' ')    /* LIB        */
          DN.VCNT.3=STRIP(TKN.3,'T',' ')    /* MEMBER     */
          DN.VCNT.4=STRIP(TKN.4,'T',' ')    /* PATH       */
          DN.VCNT.5=STRIP(TKN.5,'T',' ')    /* TIME STAMP */
          IF TTIME < TKN.5 & CRE_JCL = '' THEN CRE_JCL = 'YES ('TKN.3')'
        END
       ELSE DO
          VCNT=DN.Ø + 1
          DN.Ø=VCNT
          DN.VCNT.Ø=5
          DN.VCNT.1='.#'                    /* RULE       */
          DN.VCNT.2=''                      /* LIB        */
          DN.VCNT.3=''                      /* MEMBER     */
          DN.VCNT.4=MAC_SUBSITUDE(STRIP(LN,'L','#')) /* PATH   */
          DN.VCNT.5=Ø                       /* TIME STAMP */
          REST='\'                          /* FORCE CONTINUE */
       END
    END
    UPPER DRULE TRULE
    RULE=TRULE''DRULE              /* CONSTRUCT RULE TEMPLATE     */
    IF RN.RULE='RN.RULE' | RN.RULE.Ø='' THEN RETURN
    PROC=RN.RULE.1                 /* GET PROCEDURE NAME          */
    STP_CNT=STP_CNT+1
    STP_NM='STEP'STP_CNT
QUEUE '//*TARGET:'TP 'RE CREATE:'CRE_JCL
QUEUE '//'STP_NM 'EXEC'  PROC=','  /* CREATE EXEC CARD            */
    DO I=2 TO RN.RULE.Ø            /* SCAN RULE LINES             */
      LM=RN.RULE.I                 /* GET NEXT RULE LINE          */
      UPPER LM
      WRLS=''
      SELECT
        WHEN INDEX(LM,'$M'DRULE)>Ø THEN WRLS='M'
        WHEN INDEX(LM,'$L'DRULE)>Ø THEN WRLS='L'
        WHEN INDEX(LM,'$P'DRULE)>Ø THEN WRLS='P'
        WHEN INDEX(LM,'$'DRULE)>Ø THEN WRLS=''
        OTHERWISE WRLS='X'
      END
      IF WRLS ¬='X' THEN
         IF WRLS ¬='' THEN
            LM=SUBS_DPND(VALUE('S'WRLS),'$'WRLS''DRULE,LM)
         ELSE
            LM=SUBS_DPND(VALUE('SP'),'$'WRLS''DRULE,LM)
      SELECT
        WHEN INDEX(LM,'$M'TRULE)>Ø THEN WRLS='M'
        WHEN INDEX(LM,'$L'TRULE)>Ø THEN WRLS='L'
```

```
      WHEN INDEX(LM,'$P'TRULE)>Ø THEN WRLS='P'
      WHEN INDEX(LM,'$.'TRULE)>Ø THEN WRLS=''
      OTHERWISE WRLS='X'
    END
    IF WRLS ¬='X' THEN
       IF WRLS > '' THEN
          LM=SUBS_DPND(VALUE('T'WRLS),'$'WRLS''TRULE,LM)
       ELSE
          LM=SUBS_DPND(VALUE('TP'),'$.'WRLS''TRULE,LM)
    LM=STRIP(LM,'T',' ')
    SK='$*'                      /* MACRO USAGE TEMPLATE        */
    K=INDEX(LM,SK,1)             /* ANY USE IN THIS RULE LINE   */
    IF K>Ø THEN DO
      LM=LM' '                   /* APPEND BLANK                */
      SBV=4                      /* ASSUME PATH    WILL USED    */
      IF SUBSTR(LM,K+2,1) = 'L' THEN SBV=2  /* LIB  WILL USED    */
      IF SUBSTR(LM,K+2,1) = 'M' THEN SBV=3  /* MEM  WILL USED    */
      IF SUBSTR(LM,K+2,1) = 'P' THEN SBV=4  /* PATH WILL USED    */
      K3=4                       /* START OF RULE               */
      IF SUBSTR(LM,K+2,1) = '.' THEN K3=3 /* NO COMPONENT REFERED */
      KP =INDEX(LM,',',K+K3)      /* MACRO TERMINATE WITH ,  ?   */
      KT =INDEX(LM,' ',K+K3)      /* MACRO TERMINATE WITH    ?   */
      IF KP=Ø | KT<KP THEN KP=KT
      KT =INDEX(LM,')',K+K3)      /* MACRO TERMINATE WITH  ) ?   */
      IF KP=Ø | KT<KP THEN KP=KT
      SK=SUBSTR(LM,K,KP-K)        /* MACRO NAME                  */
      RLS=SUBSTR(LM,K+K3-1,KP-K-K3+1) /* RULE NAME                  */
      VCNT=DN.Ø
      DO II=1 TO DN.Ø             /* SCAN ALL DEPEND. FOR THIS RL */
        IF DN.II.1 ¬= RLS THEN ITERATE
        DPTH=DN.II.SBV
        LX=LEFT(LM,K-1)''DPTH
        LM=LX''SUBSTR(LM,K+LENGTH(SK),LENGTH(LM)-LENGTH(SK)+1)
        LM=MAC_SUBSITUDE(STRIP(LM,'T',' '))
        QUEUE LM
      END
    END
    ELSE
      DO
        LM=MAC_SUBSITUDE(STRIP(LM,'T',' '))
        QUEUE LM
      END
   END
 RETURN
ANALYSE_TOKEN:
  ARG TKNS
  K=LASTPOS('.',TKNS)
  TKN.=''
  IF K=Ø THEN SAY TKNS "HAS NO '.' IN"
```

```
  WPTH=LEFT(TKNS,K-1)
  WRULE=RIGHT(TKNS,LENGTH(TKNS)-K+1)
  WPAD=''                    /* NO QUOTE SIGN  */
  IF LEFT(WPTH,1)="'" THEN  /* IS ABSOLUTE PATH */
    DO
      WPAD="'"               /* KEEP QUOTE SIGN  */
      WPTH=STRIP(WPTH,B,"'")
    END
  K=1
  DO WHILE K > Ø
    K=INDEX(WPTH,'($')
    IF K>Ø THEN
      DO
       WSUBS='?'
       KK=INDEX(WPTH,')')
       IF KK>K THEN DO
          MVAR=SUBSTR(WPTH,K+2,KK-K-2)
          WSUBS=VS.MVAR
          END
       IF LEFT(WSUBS,1)="'" THEN
          DO
           WPAD="'"
           WSUBS=STRIP(WSUBS,B,"'")
          END
       WPTH=LEFT(WPTH,K-1)''WSUBS''RIGHT(WPTH,LENGTH(WPTH)-KK)
      END
  END
  WMEM=''
  WLIB=WPAD''WPTH''WPAD
  K=INDEX(WPTH,'(')                  /* ANY MEMBER DEFINED */
  IF K>Ø THEN
    DO
      WLIB=LEFT(WPTH,K-1)
      KK=LENGTH(WPTH)-K-1
      IF RIGHT(WPTH,1)¬=')' THEN KK=KK+1 /* IS MISSING ) */
      WMEM=SUBSTR(WPTH,K+1,KK)
    END
  UPPER WRULE WLIB WMEM WPTH
  TKN.1=WRULE              /* RULE    */
  TKN.2=WLIB               /* LIBRARY */
  TKN.3=WMEM               /* MEMBER  */
  TKN.4=WPTH               /* PATH    */
  TKN.5=Ø                  /* LAST MODIFICATION */
  TKN.Ø=5
/*  RETURN Ø */
  IF WMEM > '' THEN
     SELECT
     WHEN LEFT(WRULE,2)='.O' THEN              /* OBJ     */
          TKN.5 = OBJ_TIME_STAMP(WLIB,WMEM)
```

```
     WHEN LEFT(WRULE,4)='.EXE' THEN           /* EXE     */
          TKN.5 = Ø                           /* ALWAYS RECREATE EXE */
     WHEN LEFT(WRULE,3)='.GO' THEN            /* EXE     */
          TKN.5 = Ø                           /* ALWAYS RECREATE EXE */
     WHEN LEFT(WRULE,4)='.DLL' THEN           /* EXE     */
          TKN.5 = Ø                           /* ALWAYS RECREATE EXE */
     WHEN LEFT(WRULE,4)='.LIB' THEN           /* LIB     */
          TKN.5 = SRC_TIME_STAMP(WLIB,WMEM)
     WHEN LEFT(WRULE,2)='.H' THEN             /* HEADERS  */
          TKN.5 = SRC_TIME_STAMP(WLIB,WMEM)
     OTHERWISE
          TKN.5 = SRC_TIME_STAMP(WLIB,WMEM)
     END
  RETURN Ø
REGISTER_DSN_TO_ISPF:
 ARG DSN
 IF LIBID.DSN.'ISPF'.Ø='' THEN   /* NOT REGISTERED */
  DO
   IF SYSDSN("'"DSN"'") ¬= 'OK' THEN RETURN '' /* NOT VALID DSN */
   ADDRESS ISPEXEC
   "LMINIT  DATAID(DSK)   DATASET('"DSN"') ENQ(SHR)"
   WI=LIBID.Ø+1
   LIBID.Ø=WI
   LIBID.WI=DSN
   LIBID.DSN.'ISPF'.Ø=1
   LIBID.DSN.'ISPF'.1=DSK
   "LMOPEN  DATAID("DSK") OPTION(INPUT)"
  END
 ELSE
  DSK=LIBID.DSN.'ISPF'.1
 RETURN DSK
OBJ_TIME_STAMP: /* FINDS OBJECT CREATION TIME FROM INSIDE OF DECK */
 ARG DSN,MEM
 DSN=STRIP(DSN,'B')
 MEM=STRIP(MEM,'B')
 DSMEM=DSN'('MEM')'
 /* CHECK IF DSN WAS REGISTERED */
 DSK=REGISTER_DSN_TO_ISPF(DSN)
 /* CHECK ANY ISPF STATISTIC THERE */
 ZLM4DATE=''
 IF DSK > '' THEN
 ADDRESS ISPEXEC "LMMFIND DATAID("DSK") MEMBER("MEM") STATS(YES)"
 ELSE
    RETURN ''
 IF ZLM4DATE='' | ZLM4DATE='ZLM4DATE' THEN
   DO
    X=OUTTRAP('V.')
    ADDRESS TSO
    "FREE FI(DSKDD)"
```

```
    V.='';V.Ø=Ø
    "ALLOC FI(DSKDD) DSN('"DSMEM"') SHR"
    STMP=''
    IF V.Ø=Ø THEN DO
     DO WHILE STMP=''
      "EXECIO 1 DISKR DSKDD (STEM LN."
      IF RC > Ø THEN LEAVE
      IF SUBSTR(LN.1,2,3) ¬= 'TXT' THEN ITERATE   /* TIME STAMP ON TXT
*/
      IF SUBSTR(LN.1,18,3) ¬= 'ØØ-' THEN ITERATE  /* MARKED WITH ØØ- */
      STMP=SUBSTR(LN.1,29,14)
     END
     "EXECIO Ø DISKR DSKDD (FINIS"
     'FREE F(DSKDD)'
      X=OUTTRAP('OFF')
     V.=''
     RETURN TOUCH_SRC(DSN,MEM,DSK,STMP)
    END
   END
 ELSE
   DO
    STR=SUBSTR(ZLM4DATE,1,4)''SUBSTR(ZLM4DATE,6,2)''SUBSTR(ZLM4DATE,9,2)
    STR=STR''SUBSTR(ZLMTIME,1,2)''SUBSTR(ZLMTIME,4,2)''ZLMSEC
   END
 SAY 'TIME:'STR   'FOR:'DSMEM
 RETURN STR
SRC_TIME_STAMP: /* FINDS SOURCE MODIFICATION TIME IF NOT SETS */
 ARG DSN,MEM
 DSMEM=DSN'('MEM')'
 DSK=REGISTER_DSN_TO_ISPF(DSN)
 IF DSK = '' THEN RETURN ''  /* INVALID DSN  */
 ADDRESS ISPEXEC "LMMFIND DATAID("DSK") MEMBER("MEM") STATS(YES)"
 IF ZLM4DATE='' | ZLM4DATE='ZLM4DATE' THEN RETURN TOUCH_SRC(DSN,MEM,DSK)
 STR=SUBSTR(ZLM4DATE,1,4)''SUBSTR(ZLM4DATE,6,2)''SUBSTR(ZLM4DATE,9,2)
 STR=STR''SUBSTR(ZLMTIME,1,2)''SUBSTR(ZLMTIME,4,2)''ZLMSEC
 SAY 'TIME:'STR   'FOR:'DSMEM
 RETURN STR
TOUCH_SRC:      /* SETS STATISTIC OF PDS MEMBER */
 ARG DSN,MEM,DSK,TMSTMP
 DSMEM=DSN'('MEM')'
 IF DSK = 'DSK' | DSK = '' THEN DO
    DSK=REGISTER_DSN_TO_ISPF(DSN)
    IF DSK='' THEN RETURN ''
 END
 IF TMSTMP¬='' & TMSTMP¬='TMSTMP' THEN
   DO
    ZLM4DATE=LEFT(TMSTMP,8)
    ZLM4DATE=INSERT('/',ZLM4DATE,6)
    ZLM4DATE=INSERT('/',ZLM4DATE,4)
```

```
   ZLMDATE=RIGHT(ZLM4DATE,8)
   ZLMTIME=SUBSTR(TMSTMP,9,4)
   ZLMTIME=INSERT(':',ZLMTIME,2)
   ZLMSEC=SUBSTR(TMSTMP,13,2)
  END
ELSE
  DO
   ZLMDATE=DATE('ORDERED')
   ZLM4DATE='2Ø'DATE('ORDERED')
   TM=TIME()
   ZLMTIME=LEFT(TM,5)
   ZLMSEC=RIGHT(TM,2)
  END
ADDRESS ISPEXEC ,
"LMMSTATS DATAID("DSK") MEMBER("MEM") VERSION(1) MODLEVEL(Ø1)" ,
  "MODDATE("ZLMDATE") MODDATE4("ZLM4DATE") MODTIME("ZLMTIME")" ,
  "CREATED("ZLMDATE") CREATED4("ZLM4DATE") CURSIZE(1) MODRECS(1)",
  "CURSIZE(1) INITSIZE(1)"
STR=SUBSTR(ZLM4DATE,1,4)''SUBSTR(ZLM4DATE,6,2)''SUBSTR(ZLM4DATE,9,2)
STR=STR''SUBSTR(ZLMTIME,1,2)''SUBSTR(ZLMTIME,4,2)''ZLMSEC
SAY 'TIME:'STR   'FOR:'DSMEM
RETURN STR
```

Tamer Tezgel
Project Leader
AKNET AS (Turkey)

Articles for inclusion in *MVS Update* can be sent to the editor, Trevor Eddolls, at trevore@xephon.com. A copy of our *Notes for Contributors* can be downloaded from www.xephon.com/nfc.

Neon Systems has announced general availability of its Shadow Console interleaved management, monitoring, and debugging tool, designed to ensure the performance and availability of composite applications that integrate application platform suites with mainframe data and transactions.

Through its Windows-based GUI, it offers application developers and production operations staff a consolidated end-to-end view of the middleware component from the initial API adapter call in the application platform suite to the back-end data or transaction source on the mainframe.

Because it uses a modern GUI, users don't need to be familiar with legacy user interface software like TSO/ISPF. It doesn't require a mainframe user ID and password for authorized customer personnel to view detailed consolidated traces of application request activity.

Using diagnostic and control facilities inherent in other Shadow products, it consolidates information from the Shadow Client interfaces and data from the Shadow Server, which resides on z/OS.

It allows both mainframe and non-mainframe users to identify and resolve problems that may be affecting the performance and availability of an application, both during the development cycle and when the application is deployed in production environments.

Other features include the ability to provide monitoring. System alerts and exceptions are detected and forwarded via SNMP to an existing network management system for integration with enterprise application, computer, and network status information to provide a broad view of application execution.

The software can be deployed with leading J2EE and .NET application platform suites to provide customers with J2CA, JDBC, or ODBC access to mainframe data sources and transaction environments, supporting DB2, CICS/TS, IMS/TM, IMS/DB, VSAM, ADABAS, Natural/ACI, flat files, IDMS, and other z/OS data and transactional sources.

For further information contact:
NEON Systems, 14100 Southwest Freeway, Suite 500, Sugarland, TX 77478, USA.
Tel: (281) 491 4200.
URL: http://www.neonsys.com/solutions/shadow.

* * *

Serena Software has integrated its ChangeMan DS software change manager for distributed systems with the TeamTrack defect and issue management system from TeamShare.

The combination, we're told, provides TeamTrack users with an automated change management system that helps streamline software development and improve communication across the enterprise. In addition, the integration also allows joint customers to integrate with other vendors' tools.

For further information contact:
Serena Software, 2755 Campus Drive, 3rd Floor. San Mateo, CA 94403, USA.
Tel: (650) 522 6600
URL: http://www.serena.com.