182 MVS update

November 2001

In this issue

Published by
Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 33598
From USA: 01144 1635 33598
E-mail: Jaimek@xephon.com

North American office
Xephon/QNA
PO Box 350100,
Westminster, CO 80035-0100
USA
Telephone: (303) 410 9344
Fax: (303) 438 0290

Contributions
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, you can download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

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
Jaime Kaminski

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.

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 1992 issue, are available separately to subscribers for £29.00 ($43.50) each including postage.

Printed in England.

LPA module mapping

As the LPA becomes more dynamic and as more products hook themselves into the system, it is not always clear what modules are in the 'classic' LPA and which have been dynamically loaded. As part of an exercise to try to understand what was where, I decided to have a go at using the CSVINFO macro to map the contents of the LPA. Once I had got that working I used the addresses that this macro returned as a feed into the REXX from the *Memory mapping* article in *MVS Update* Issue 181, October 2001, to quickly identify where the modules were in the system. This was then displayed as a simple ISPF table (an example of the first screen of which now follows):

```
LPA module details for LPAR PRD1    Row 1 to 20 of 2,198
Command ===>                                          Scroll ===> PAGE

    Module    Amode    Length     Load      Entry     Location
   IGDERRC2   31     1592.13K    0B18B000  0B256018  E-PLPA
   BPXGYWRT   31      403.73K    09ED1000  09F1C130  E-PLPA
   BPXGYFLT   31      403.73K    09ED1000  09F192C8  E-PLPA
   CAS9SAFC   31        8.62K    00C3ED88  00C3ED88  CSA
   CAS9SEC    24        1.52K    00C4C140  00C4C140  CSA
   CASMINIT   31        1.80K    00C418D0  00C418D0  CSA
   CA£ULPA4   31        2.50K    13C7D100  13C7D100  E-CSA
   TMSUX2S    24        0.95K    00C9C068  00C9C068  CSA
   TMSUX2C    31        0.51K    13F84448  13F84448  E-CSA
   TMSUX2B    31        1.09K    13F7C048  13F7C048  E-CSA
   TMSUX2A    31        2.07K    13CE5468  13CE5468  E-CSA
   CTSOCE00   24        2.22K    00C93080  00C93080  CSA
   TMS0MODV   31        4.02K    13A24FF0  13A24FF0  E-CSA
   TMSSB01    31        2.90K    13CEF1D0  13CEF1D0  E-CSA
   TMSQSTS    24        2.77K    00C8A1F0  00C8A1F0  CSA
   CTSPM      31       11.00K    00C46000  00C46000  CSA
   TMSPM      31       11.00K    00C46000  00C46000  CSA
   CTSMSGVT   31        2.67K    13E90150  13E90150  E-CSA
   CTSMSGSP   31        1.77K    13F2C8F0  13F2C8F0  E-CSA
   CTSMSGRT   31        5.83K    13A298B0  13A298B0  E-CSA
```

This dialog also uses the storage display REXX from the Memory mapping article in *MVS Update* Issue 181 to enable a dump view of the module. You can use the dialog to view either the module load point (by issuing ZL alongside the table entry) or the entry point (through a ZE line command). An example is shown below:

```
STORAGE DISPLAY             Row 1 to 19 of 64

Command ===>                                                Scroll ===> PAGE

         Ø        4        8        C
======== |─── |─── |─── |───    ================ Region
ØB256Ø18 47F0F026 20C9C7C4 C5D9D9C3 F2F0F261    .ØØ..IGDERRC2Ø2/ E-PLPA
ØB256Ø28 F1F261F9 F3C8C4E9 F1F1C3FØ 4ØD6E8F4    12/93HDZ11CØ OY4 E-PLPA
ØB256Ø38 F1F7F3F1 4ØØØ9ØEC DØØC18CF 41A0CFFF    1731  ..}....... E-PLPA
ØB256Ø48 1FFF43F0 A3711FØØ BFØ7A372 5Ø8ØDØØ8    ...ØT.. ..T.&.}  E-PLPA
ØB256Ø58 47F0C048 ØB1F9EFA 58F0C044 Ø5EF588Ø    .Ø{......Ø{..... E-PLPA
ØB256Ø68 DØØ818B1 4190BFFF 5ØDØB004 5ØB0DØØ8    } ......&}..&.}  E-PLPA
ØB256Ø78 98F1DØ1Ø 18DBD2Ø7 BØ481ØØØ 47F0C092    Q1}...K.... .Ø{K E-PLPA
ØB256Ø88 2ØC9C7C4 C5D9D9C3 F2FØF761 F1F561F9    .IGDERRC2Ø7/15/9 E-PLPA
ØB256Ø98 F5C8C4E9 F1F1C3F0 4ØD5D6D5 C5404040    5HDZ11CØ NONE    E-PLPA
ØB2560A8 4ØØØD221 B190A137 4110B19Ø 58F0A061     K...¯......Ø./  E-PLPA
ØB2560B8 Ø5EF1F88 5Ø80B158 5Ø80B15C 45E0C2AE    ...H&...&..*.\B. E-PLPA
ØB2560C8 5880BØ48 58108ØØØ 4100B248 5ØØØ10EC    ....... . ..& .. E-PLPA
ØB2560D8 4110A1A2 5010B0E4 5Ø80B0E8 58800010    ..¯S&..U&..Y.. . E-PLPA
ØB2560E8 58808128 58808064 58808054 58808020    ..A............. E-PLPA
ØB2560F8 4180800C 5Ø80B0EC 4110B0E4 45E0C3Ø6    ....&......U.\C. E-PLPA
ØB256108 9201B1B8 D702B1B9 B1B9588Ø BØ485880    K...P........... E-PLPA
ØB256118 80005880 80EC5080 B1BCD707 B1C0B1C0    . ....&...P..{.{ E-PLPA
ØB256128 4110B1B8 B22700F0 90FCD010 58800010    ...... Ø..}... . E-PLPA
ØB256138 58E08304 5820E094 B2182000 5820D010    .\C...\M... ..}. E-PLPA
```

All that is required to enable this dialog in conjunction with the previous one is the assembly of one piece of code (which does not require any special linkage) into your STEPLIB, the installation of two help panels into your ISPPLIB and one REXX into your SYSPROC concatenation; plus, of course, all the code from the Memory mapping article. Then simply invoke the dialog by issuing the command TSO CSVLPAR1, which is the name given to the REXX in this document.

Please note that the macros for this code can be obtained from *MVS Update* Issue 181, October 2001.

CSVLPA

```
/*********************************************************************/
* CSVLPA: A REXX FUNCTION TO LIST ALL THE LPA MODULES
* USAGE: CALL CSVLPA
* NOTE: CSVLPA RETURNS AN ARRAY OF MODULE NAMES
*        CSV_MODULE ........... LPA MODULE NAME
*        CSV_MODULE_AMODE .... ADDRESSING MODE 31-BIT OR 24-BIT
*        CSV_MODULE_LENGTH ... LENGTH OF MODULE
*        CSV_MODULE.LOAD_POINT ... ADDRESS WHERE LOADED
*        CSV_MODULE.ENTRY_POINT ... MODULE ENTRY POINT
```

```
*********************************************************************
CSVLPA TITLE 'REXX FUNCTION TO RETRIEVE LNK AND LPA INFO'
CSVLPA AMODE 31
CSVLPA RMODE ANY
CSVLPA CSECT
       REXREGS
         BAKR 14,Ø
         LR   12,15
         USING CSVLPA,12
         LR    R1Ø,RØ                  *R1Ø -> A(ENVIRONMENT BLOCK)
         USING ENVBLOCK,R1Ø
         LR    R11,R1                  * R11 -> A(PARAM LIST (EFPL))
         USING EFPL,R11
         L     R9,ENVBLOCK_IRXEXTE     *R9 -> A(EXTERNAL EP TABLE)
         USING IRXEXTE,R9
*
* GET A WORK AREA FOR REXX OUTPUT
* MAP WITH R2 ... NEED TO DO THIS BEFORE ANY ROUTING TO POSSIBLE
* REXX VARIABLE OUTPUT (EG ROUTINE ABENDØØ1)
*
         STORAGE OBTAIN,LENGTH=AREALEN,ADDR=(2)
         USING WORKAREA,2
*
* PREPARE THE REXX AREA FOR USE
*
         XC  COMS(COMSLEN),COMS * SET TO LOW VALUES
         LA  15,COMID
         ST  15,COMS
         LA  15,COMDUMMY
         ST  15,COMS+4
         ST  15,COMS+8
         LA  15,COMSHVB
         ST  15,COMS+12
         LA  15,COMRET
         ST  15,COMS+16
         OI  COMS+16,X'8Ø'
         MVC COMID,=C'IRXEXCOM'
*
* OBTAIN ALL THE MODULE INFORMATION
*
        STM  2,12,MYREGS
        CSVINFO FUNC=LPA,ENV=MVS,MIPR=CSVMIPR
        B RETURN
*
* THE FOLLOWING IS THE MIPR ROUTINE FOR RETRIEVING THE LPA
* MODULE NAME LIST.
CSVMIPR DS ØH
        BAKR 14,Ø * NEXT LAYER OF REGISTER SAVES
        USING CSVMIPR,15
        LM   2,12,MYREGS
        DROP 15
*
```

```
* RESET THE REGISTER MAPPINGS BACK TO THOSE OF THE MAIN MODULE
*
         USING CSVLPA,12
         LR   R7,R1
         USING MODI_HEADER,R7
         L    R8,MODI_1_PTR
         USING MODI_1,R8
         L    R3,MODI_2_PTR
         USING MODI_2,R3
         TM   MODI_ENT@,X'8Ø'  * IS IT A 31 BIT MODULE?
         BC   1,ITS_31         * YES SO SET APPROPRIATE VARIABLE
         SHOWARAY MOD24,CSV_MODULE_AMODE
         B    SHOW_NAME
*
ITS_31   DS ØH
         SHOWARAY MOD31,CSV_MODULE_AMODE
*
SHOW_NAME DS ØH
*
         SHOWARAY MODI_8_BYTE_NAME,CSV_MODULE
         SHOWARAY MODI_MOD_LEN,CSV_MODULE_LENGTH,DEBIN=4
         SHOWARAY MODI_LOAD@,CSV_MODULE_LOAD_POINT
         MVC MOD_POINT,MODI_ENT@     * CLEAR HI ORDER BIT FOR CLARITY
         NI  MOD_POINT,X'7F'
         SHOWARAY MOD_POINT,CSV_MODULE_ENTRY_POINT
         XR 15,15
         PR
RETURN   DS ØH
*** ONCE ALL THE MODULE DETAILS HAVE BEEN OUTPUT SET UP THE BASE
         SHOWBASE CSV_MODULE * PUT OUT TOTAL NUMBER OF ENTRIES
*********************************************************************
***      RETURN TO CALLER
***      RELEASING ALL STORAGE IN THE PROCESS
*********************************************************************
         STORAGE RELEASE,LENGTH=AREALEN,ADDR=(2)
         PR
*********************************************************************
***      WORKING STORAGE ETC                                      ***
*********************************************************************
         TITLE  'WORKING STORAGE / DSECTS'
         LTORG
MOD24    DC C'24'
MOD31    DC C'31'
MYREGS   DS  7D
*
WORKAREA DSECT
*
*      IRXEXCOM PARAMETER AREA
*
         DS  ØD
COMS     DS  5AL4
```

```
COMID    DS  CL8
COMDUMMY DS  AL4              * NOT USED
COMSHVB  DS  (SHVBLEN)X       * IRXEXCOM SHVBLOCK (LENGTH FROM DSECT)
COMRET   DS  AL4              * IRXECOM RC
         DS  ØD
COMSLEN  EQU *-COMS
MOD_POINT DS CL4
AREALEN  EQU *-COMS
         CSVMODI
         IRXEFPL
         IRXARGTB
         IRXEVALB
         IRXENVB
         IRXEXTE
         IRXSHVB
         END
```

CSVLPAR1

```
/* REXX */
/* */
/* Display LPA module information. */
/* */
/* */
/* Obtain the LPAR name for the panel display */
/* */
CVTECVT=D2X(C2D(STORAGE(1Ø,4))+14Ø) /* point to cvtsysad */
lparname=STRIP(STORAGE(D2X(C2D(STORAGE(CVTECVT,4))+344),8))
/* */
restart:
rowpos=1
CALL CSVLPA    /* get the module information */
CALL RMAPSTOR  /* and get storage mapping information. */
address ispexec
/* */
/* The table entry l1 is kept to allow easy numeric sorting */
/* */
'TBCREATE LPAMOD NAMES(module amode length l1 loadpt entrypt location),
 NOWRITE REPLACE'
/* */
/* Now loop around to create the table */
/* */
/* */
DO x=1 to csv_module.Ø
/* */
CALL location_find C2X(csv_module_load_point.x)
location=result
module=csv_module.x
loadpt=C2X(csv_module_load_point.x)
entrypt=C2X(csv_module_entry_point.x)
```

```
amode=csv_module_amode.x
length=FORMAT((csv_module_length.x/1Ø24),,2)||'K'
l1=csv_module_length.x*1
'TBADD LPAMOD'
END
redisplay:
'TBTOP LPAMOD'
'TBSKIP LPAMOD NUMBER('rowpos')'
'TBDISPL LPAMOD PANEL(CSVLPAP1)'
rowpos=ztdtop
/* */
/* User command processing */
/* */
IF zcmd='REFRESH' THEN SIGNAL restart
IF reply='END' THEN EXIT
IF WORD(zcmd,1)='L' THEN CALL find_process
IF WORD(zcmd,1)='SORT' THEN CALL sort_process
IF ztdsels\=Ø THEN DO
   CALL table_routine
   selector=''
   SIGNAL redisplay
   END
IF reply='ENTER' THEN SIGNAL redisplay
/* */
/* now process the selection commands */
/* */
table_routine:
DO prime=1 to ztdsels
   UPPER selector
   selector.prime=selector
   load.prime=loadpt
   entry.prime=entrypt
   IF ztdsels =1 THEN LEAVE
   IF ztdsels >1 THEN 'TBDISPL LPAMOD'
END
DO x=1 TO prime
   IF selector.x='ZL' THEN ADDRESS TSO '%STORDISR' load.x
   IF selector.x='ZE' THEN ADDRESS TSO '%STORDISR' entry.x
END
RETURN
/* */
/* Retrieve the storage areas */
/* */
location_find:
arg testaddr
testaddr=X2C(RIGHT('ØØØØØØØØ'||testaddr,8))
location='Unknown Address'
DO y=1 TO map_name.Ø
   IF testaddr>=start_address.y & testaddr<=end_address.y
   THEN LEAVE
END
```

```
location=map_name.y
RETURN location
/* */
/* this part of the REXX is used to locate the requested dataset */
/* */
find_process:
'TBVCLEAR LPAMOD'
module=WORD(zcmd,2)
'TBSCAN LPAMOD NEXT ARGLIST(module) POSITION(rowpos)'
IF RC¬=Ø THEN DO
  'TBSCAN LPAMOD PREVIOUS ARGLIST(module) POSITION(rowpos)'
   IF RC¬=Ø THEN DO
      zedsmsg=module 'not found'
      zedlmsg='Try a different search'
      ADDRESS ISPEXEC 'SETMSG MSG(ISRZØØ1)'
     END
  END
RETURN
sort_process:
rowpos=1      /* reset the table positioning */
IF WORDS(zcmd)=1 THEN DO
   COL='MODULE'
   order='A'  /* set default sort to ADDRESS ascending */
   END
ELSE IF WORDS(zcmd)=2 THEN DO
     col=WORD(zcmd,2)
     order='A'  /* set default order for sorting */
     END
ELSE IF WORDS(zcmd)=3 THEN DO
     col=WORD(zcmd,2)
     order=WORD(zcmd,3)
     END
ELSE DO
     zedsmsg='SORT command error'
     zedlmsg='Too many parameters supplied'
     'SETMSG MSG(ISRZØØ1)'
     RETURN
     END
IF order='A' | order='D' THEN DO
  IF col='MODULE' THEN 'TBSORT LPAMOD FIELDS(module,c,'order')'
  ELSE IF col='AMODE' THEN 'TBSORT LPAMOD FIELDS(amode,n,'order')'
  ELSE IF col='LENGTH' THEN 'TBSORT LPAMOD FIELDS(l1,n,'order')'
  ELSE IF col='LOAD' THEN 'TBSORT LPAMOD FIELDS(loadpt,c,'order')'
  ELSE IF col='ENTRY' THEN 'TBSORT LPAMOD FIELDS(entrypt,c,'order')'
  ELSE IF col='LOCATION' THEN 'TBSORT LPAMOD FIELDS(location,c,'order')'
  ELSE DO
       zedsmsg='SORT command error'
       zedlmsg='Unknown column' col 'specified'
      'SETMSG MSG(ISRZØØ1)'
  END
END
```

```
ELSE DO
     zedsmsg='SORT command error'
     zedlmsg='third parameter must be A or D'
     'SETMSG MSG(ISRZØØ1)'
END
RETURN
```

CSVLPAP1

```
)Attr Default(%+_)
   | type(output) intens(high) caps(on ) just(left )
   > type(output) intens(high) caps(on ) just(right)
   @ type(output) intens(low ) caps(off) just(asis )
)Body  Expand(//)
/ /% LPA module details for LPAR!lparname / /
%Command ===>_zcmd                                / /%Scroll ===>_amt +
+
    Module    Amode    Length     Load      Entry     Location
)Model
_z |z         |z    >z        +  !z        !z        !z
)Init
  .Help = csvlpah1               /* insert name of tutorial panel  */
  .ZVARS = '(selector module amode length loadpt entrypt location)'
  &amt = PAGE
)PROC
&REPLY = .RESP
)End
```

CSVLPAH1

```
)ATTR
 ' TYPE(PT)                        /* panel title line            */
 ? TYPE(PIN)                       /* panel instruction line      */
 # TYPE(NT)                        /* normal text attribute       */
 } TYPE(ET)                        /* emphasized text attribute   */
 [ TYPE(DT)                        /* description text            */
 | AREA(SCRL)                      /* scrollable area attribute   */
)BODY
'──────── Help panel for LPA Module Dialog ──────────-
+
+Command ==>_ZCMD        +
+
+This panel provides information on the modules which are part of the
+LPA.
+
+=============================================================================
|pnarea                                                                      |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
```

```
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
|                                                                              |
+==============================================================================
+
%Use ENTER to scroll downwards through the available data.
)AREA pnarea
#
}DESCRIPTION:
+
+This screen displays a variety of information about the LPA
+modules. Six columns are shown as follows:
+
+Column 1: the name of the module in storage.
+
+Column 2: this is the addressing mode of the module.
+
+Column 3: the size of the module in 'K'.
+
+Column 4: the address in memory where this module is loaded.
+
+Column 5: the entry point address of the module.
+
+Column 6: this identifies where in system storage the module's
+          load point corresponds to (LPA, CSA etc)
+
#
}LINE COMMANDS:
+
+ZL .... this will display the storage at the load point of the
+        module.
+
+ZE .... this will display the storage at the entry point of the
+        module.
#
}SUBCOMMANDS:
+
+REFRESH: This will cause a rebuild of the table. Use this if the
+         LPA has been modified and you wish to check the changes.
+
+SORT:    Use this command to sort the module list into an order.
+         By default the order is to put the module names into
+         ascending order. If you wish to sort in a different order
+         or you wish to select another column then the command is
+         SORT column_head A/D where an A for ascending is assumed
+         if not explicitly specified. Hence to get a list in module
+         size order with the largest module at the top, issue:
+         SORT LENGTH D
```

```
+
+L:       Using the L subcommand allows a search for a specific
+         module. Wild carding is acceptable using an * as the last
+         character.
)PROC
&ZTOP=CSVLPAH1
&ZUP=CSVLPAH1
&ZCONT=CSVLPAH1
)END
```


Useful freeware

The PC is a ubiquitous tool these days, even for the mainframe specialist. However, the usual corporate PC does not always have everything that we mainframe people would like to have. As a result it can often be worth a trawl of the freeware sites for useful add-ons. The following list is a collection of three freeware programs that I found at: *http://www.zdnet.co.uk* that I have found very useful:

- *Netpad* – this program permits the creation of network diagrams which can be exported as bit maps and included into a word processing package. Very handy for describing links between systems.
- *PC Magazine's MultiRen* – this package is very useful if you download multiple PDS members onto a PC. It allows users to rename multiple files that you have selected by right-clicking and choosing the multiple rename. The really neat feature is it allows you to add file extensions and to switch the case of file names. Hence you can download a PDS and then add .txt to each entry and make it easy to read on your PC (for example).
- *123 Password Recovery* – this little program is a useful add-on. If like me you have used the save password feature of Windows for all your sites that you have registered for, then you have probably forgotten some of those passwords. This program turns the asterisks in the password field back into clear text.

A cheap Auxiliary Storage Monitor

INTRODUCTION

Auxiliary storage shortage tends to be critical on MVS machines, and it is not easy to anticipate, specially if the workload is a mix of heavy batch jobs, DB2 transactions, or if you are working on a test or development machine with a chaotic workload. Unfortunately, IBM does not give you a warning until 70% of all available slots in the system are in use, ie:

```
IRA200E AUXILIARY STORAGE SHORTAGE
```

IBM also gives you the IRA203I message to identify the address spaces with the most rapidly increasing auxiliary storage requirements. But in some situations this can be too late.

Before the critical 70%, the only tool that you have to estimate what is going on with auxiliary storage is the D ASM MVS command.

This program will act like a watchdog for your auxiliary storage, sending you some warnings before the critical 70% of all available slots in the system are in use. With a bit of practice, you will be able to know what is 'normal' for your site, and to fix your own thresholds to trigger specific alerts.

It sends WTOs to inform about the amount of auxiliary storage usage, and will give you the names of the address spaces occupying more than 1% of the total. This program will act differently if you execute it under TSO or in batch/STC mode:

- Under TSO, it will scan the address space vector table once, give you the information, and finish.
- In batch/STC mode, it will stay on the machine until you cancel it and scan the address space vector table every minute.

It will send WTOs at least every 20 minutes if everything is fine from the auxiliary storage usage point of view, or every minute otherwise. For the program, things are going wrong when the total auxiliary storage usage is rising 40%, or at least one address space is occupying more than 15% of the total slots.

The program will give you the following information every 20 minutes or every minute:

- Total auxiliary storage in megabytes
- Total auxiliary storage usage in megabytes
- Total auxiliary storage usage in percent
- For every address space occupying more then 1%
  - its name
  - its occupancy in percent
  - its occupancy in megabytes.

IMPLEMENTATION

Assemble and link this program in a ‘normal’ Loadlib. No special authorization is required because all the control blocks that we need are in common area. Please note that this program uses two ‘in-house’ macros:

- INITL to start the program (and get some memory for the save area, chaining of save areas, and register equates).
- RCNTL at the end of the program (which restores registers, frees save area, and returns).

You can substitute these with your own.

PUAUXMON

```
PUAUXMON CSECT
PUAUXMON AMODE 31
PUAUXMON RMODE ANY
*********************************************************************
* Use this program to monitor auxiliary storage usage.
*********************************************************************
*
* Example of WTOs sent by this program :
*
* +PUAUXMON Total    Aux.    Storage     2250 Megs.
* +PUAUXMON AdSpace XX123AB    1 %         26 Megs.
* +PUAUXMON AdSpace XX2585d    9 %        221 Megs.
```

```
* +PUAUXMON AdSpace XX852D2    1 %         24 Megs.
* +PUAUXMON AdSpace XX1475F    1 %         3Ø Megs.
* +PUAUXMON AdSpace XX529PO   1Ø %        245 Megs.
* +PUAUXMON AdSpace XX357D5T   1 %         27 Megs.
* +PUAUXMON AdSpace XX25TG78   1 %         4Ø Megs.
* +PUAUXMON AdSpace XX95B7YH   4 %        1Ø5 Megs.
* +PUAUXMON Occupied Aux.    Storage      944 Megs.
* +PUAUXMON Occupied Aux.    Storage       41 %.
*
********************************************************************
* You can change the thresholds to trigger the WTOs by changing the
* following variables in this program :
*
* - USGTOT : threshold to consider the total of auxiliary storage
*            usage fine or not (4Ø% of total aux stor in my case)
*
* - USGONE : threshold to retain an address space to be present
*            in the display or not (1% of total aux stor in my case)
*
* - USGTWO : threshold to consider the situation going 'abnormal'
*            for one address space (15% of total aux stor in my case)
*
* - MSGTRIG: maximum of minutes without sending WTOs (2Ø minutes
*            in my case).
*
* - WAITINTV: interval between two scans (1 minute in my case).
*
********************************************************************
*
* Logic of this program :
*
* - Housekeeping
* - Checks if we are under TSO                  CHECK_TSO
* - Get info from ASVT                          GET_ASVT
* - Send the messages if any                    SEND_MSG
* - If TSO, return                              RETURN
* - If not, wait 1 minute and loop to GET_ASVT
*
********************************************************************
*
* INPUT  : Nothing
* OUTPUT : Some WTOs to indicate who is occupying the Auxiliary
*          storage.
*
********************************************************************
         EJECT
********************************************************************
* Return codes :                                                   *
*                                                                  *
* Ø  : Everything is fine.                                         *
```

```
*                                                                     *
***********************************************************************
* Conventions :                                                       *
*                                                                     *
* # prefixed fields are flags                                        *
***********************************************************************
* REGISTER USAGE                                                      *
*                                                                     *
* RØ  : Reserved                                                      *
* R1  : Reserved for macros                                           *
* R2  : Reserved for TRT instruction                                  *
* R3  : First base register                                           *
* R4  : Second base register                                          *
* R5  : Not used                                                      *
* R6  : Not used                                                      *
* R7  : Work register                                                 *
* R8  : Work register                                                 *
* R9  : Work register                                                 *
* R1Ø : Work register                                                 *
* R11 : Work register                                                 *
* R12 : Work register                                                 *
* R13 : Reserved as savearea pointer                                  *
* R14 : Reserved as link register (return address)                    *
* R15 : Reserved for return code                                      *
***********************************************************************
* Lked attributes:                                                    *
* AMODE 31                                                            *
* RMODE ANY                                                           *
* AC(Ø)                                                               *
***********************************************************************
* How to execute this program:                                        *
*                                                                     *
* - under TSO just type 'TSO PUAUXMON'                                *
*   (assuming that the program is accessible through the linklist     *
*    and you have WTPMSG in your TSO PROFILE)                         *
*                                                                     *
* - in batch/STC mode                                                 *
*   //PUAUXMON EXEC PGM=PUAUXMON                                     *
*   //STEPLIB  DD   DISP=SHR,DSN=my.load       (if not in linklist)  *
*   //SYSUDUMP DD   SYSOUT=X                                          *
*                                                                     *
***********************************************************************
         EJECT
*
***********************************************************************
* Some housekeeping. R3 and R4 are base registers.                    *
***********************************************************************
*
         INITL 3,4,EQU=R
         EJECT
```

```
*
*********************************************************************
* Main logic.                                                       *
*********************************************************************
*
         BAS   R14,CHECK_TSO           Let see if we are under TSO
LOOP     BAS   R14,GET_ASVT            Get info from ASVT
         BAS   R14,SEND_MSG            Send the messages.
         TM    #PGMFLAG,#TSO           Are we under TSO?
         BO    RETURN                  Yes, return
         STIMER WAIT,DINTVL=WAITINTV   No, wait 1 minute
         B     LOOP                    And loop
         EJECT
*
*********************************************************************
* This routine checks if we are under TSO. If so we just put a flag *
* on, to remember that when we have to decide if we do a second     *
* pass through the program or if we go out.                         *
*********************************************************************
*
CHECK_TSO      DS ØH
         BAKR  R14,Ø                   PUSH ENVIRONMENT INTO STACK
         LA    R12,Ø                   GET PSA addr
         USING PSA,R12                 ESTABLISH ADDRESSABILITY
         L     R12,PSAAOLD             GET ADDR OF CURRENT addr SPACE
         USING ASCB,R12                ESTABLISH ADDRESSABILITY
         L     R7,ASCBTSB              ADDRESS SPACE TSO?
         LTR   R7,R7
         BZ    PR321654
         OI    #PGMFLAG,#TSO           Yes, a flag to remember
*
PR321654 PR                            POP STACK AND RETURN TO CALLER
         EJECT
*
*********************************************************************
* This routine gets the info from the ASVT (Address Space Vector    *
* Table) and triggers the ASVT scan.                                *
*********************************************************************
*
GET_ASVT DS    ØH
         BAKR  R14,Ø                   PUSH ENVIRONMENT INTO STACK
         BAS   R14,INIT_FIELDS         Reinit some fields
         LA    R12,Ø                   GET PSA addr
         USING PSA,R12                 ESTABLISH ADDRESSABILITY
         L     R12,FLCCVT              GET CVT addr
         S     R12,=F'256'             GO BACK TO CVT PREFIX AREA
         USING CVTFIX,R12              ESTABLISH ADDRESSABILITY
         MVC   ASVTà,CVTASVT
         L     R11,CVTASMVT            Let see the ASMVT
         L     R11,112(R11)            Total number of slots
```

```
         ST    R11,MAXSLOTH            Just keep it
         ST    R11,ADDRESS1
         BAS   R14,CONVERT_TO_MEG      Convert to megabytes
         MVC   MSG1+34(8),ADDRESS2     Populate the WTO
         L     R8,CURWTO               Keep the wto
         MVC   Ø(LMSG,8),MSG1          In the message table
         LA    R8,LMSG(R8)             Refresh message table pointer
         ST    R8,CURWTO
         L     R12,CVTASVT             Get ASVT address
         USING ASVT,R12                Establish addressability
         BAS   R14,ASVT_SCAN           Loop in the ASVT
*
* Calculate the total of occupied slots
*
         L     R9,VIOSH                Calculate total
         A     R9,NVIOSH
         ST    R9,ADDRESS1
         BAS   R14,CONVERT_TO_MEG      Convert to megabytes
         MVC   MSG2+34(8),ADDRESS2
         BAS   R14,CALCUL_PORCENT      Calculate % used
         MVC   MSG3+39(3),ADDRESS2+5
         CLC   ADDRESS2+5(3),USGTOT    If total % used  > 4Ø %
         BL    MOVEMSG                 Trigger the send
         MVC   SENDMSG,=C'YES'
MOVEMSG  DS    ØH
         L     R8,CURWTO               Keep the 1st end message
         MVC   Ø(LMSG,8),MSG2          In the message table.
         LA    R8,LMSG(R8)             Rrefresh the table msg pointer.
         ST    R8,CURWTO
         L     R8,CURWTO               Keep the 2nd end message
         MVC   Ø(LMSG,8),MSG3          In the message table.
         LA    R8,LMSG(R8)             Rrefresh the table msg pointer.
         ST    R8,CURWTO
         PR                            POP STACK AND RETURN TO CALLER
         EJECT
*
*********************************************************************
* This routine inits some fields.                                   *
*********************************************************************
*
INIT_FIELDS    DS ØH
         BAKR  R14,Ø                   PUSH ENVIRONMENT INTO STACK
         MVC   VIOSH,=F'Ø'             Total VIO
         MVC   NVIOSH,=F'Ø'            Total NO VIO
         LA    R8,WTOTAB               Beginning of the table message
         ST    R8,CURWTO
         MVC   SENDMSG,=C'NO '         Init flag to NO
         PR                            POP STACK AND RETURN TO CALLER
         EJECT
*
```

```
*
*********************************************************************
* This routine scans the ASVT entry by entry and extracts information
*********************************************************************
*
ASVT_SCAN      DS ØH
         BAKR  R14,Ø                    PUSH ENVIRONMENT INTO STACK
         L     R11,ASVTMAXU             COUNTER FOR LOOP
         LA    R9,ASVTENTY              GET FIRST ENTRY
         ST    R9,MASTASVT              SAVE MASTER ASVT ENTRY addr
         OI    MASTASVT,àHIGHON         HIGH BIT ON
ASCBLOOP DS    ØH
         CLC   Ø(4,R9),MASTASVT         UNAVAILABLE ENTRY?
         BE    RUNLOOP                  Yes, see the next one
CHKVALID DS    ØH
         TM    Ø(R9),ASVTRSAV           VALID ASCB
         BO    RUNLOOP                  NO, CHECK NEXT ASVT ENTRY
         AP    ASPNUM,ONE               +1 TO NUMBER OF CURRENT adSPACE
         BAS   R14,ASCB_SCAN            Let see this ASCB
RUNLOOP  DS    ØH
         LA    R9,4(,R9)                NEXT ASVT ENTRY
         BCT   R11,ASCBLOOP             CONTINUE TILL ASVTMAXU REACHED
         PR                             POP STACK AND RETURN TO CALLER
         DROP  R12
         EJECT
*
*********************************************************************
* This routine process one ASCB.                                    *
*********************************************************************
*
ASCB_SCAN      DS ØH
         BAKR  R14,Ø                    PUSH ENVIRONMENT INTO STACK
         L     R8,Ø(R9)                 GET ASCB addr
         USING ASCB,R8                  ESTABLISH ADDRESSABILITY
         L     R7,ASCBJBNS              GET JOB NAME FOR STC
         MVC   JOBNAME,Ø(R7)
CHKTSO   DS    ØH
         L     R7,ASCBTSB               GET TSB
         LTR   R7,R7                    IS ZERO?
         BNZ   CHKASSB                  No, let see the ASSB
CHKBATCH DS    ØH
         L     R7,ASCBJBNI              GET addr INITIATED JOBNAME
         LTR   R7,R7                    IS ZERO?
         BZ    CHKASSB                  YES, IT'S STC OR MOUNT AdSpace
         L     R7,ASCBJBNI              GET JOB NAME FOR BATCH
         MVC   JOBNAME,Ø(R7)
CHKASSB  DS    ØH
*
* Let's see auxiliary storage usage for this address space.
*
```

```
         BAS   R14,GET_AUX_INFO
*
         PR                            POP STACK AND RETURN TO CALLER
         DROP  R8
         EJECT
*
*
*********************************************************************
* This routine calculates the auxiliary storage usage for one       *
* address space.                                                    *
* When we enter in this routine R8 contains the ASCB address.       *
*********************************************************************
*
GET_AUX_INFO   DS ØH
         BAKR  R14,Ø                   PUSH ENVIRONMENT INTO STACK
         USING ASCB,R8                 ESTABLISH ADDRESSABILITY
         L     R8,ASCBASSB             GET ASSB addr
         USING ASSB,R8                 ESTABLISH ADDRESSABILITY
         L     R9,ASSBNVSC             NUM SLOT NO VIO
         A     R9,NVIOSH
         ST    R9,NVIOSH
         L     R7,ASSBVSC              NUM SLOT    VIO
         A     R7,VIOSH
         ST    R7,VIOSH
         MVC   SLOTVIOH,ASSBVSC        NUM SLOT    VIO
         L     R9,ASSBNVSC             NUM SLOT NO VIO
         A     R9,ASSBVSC              NUM SLOT    VIO
         ST    R9,SLOTTOTH             Total slots
         ST    R9,ADDRESS1
         BAS   R14,CONVERT_TO_MEG      Convert to megabytes
         MVC   MEGTOT,ADDRESS2
         BAS   R14,CALCUL_PORCENT      Calculate % usage
         MVC   PCTOCU,ADDRESS2+5
         CLC   PCTOCU,USGONE           % usage >= 1,
         BL    PR123                   Keep it.
         CLC   PCTOCU,USGTWO           % usage > 14,
         BL    MOVE1                   triggers the message.
         MVC   SENDMSG,=C'YES'
MOVE1    DS    ØH
         MVC   MSG4+17(8),JOBNAME      Keep
         MVC   MSG4+26(3),PCTOCU       the
         MVC   MSG4+34(8),MEGTOT       info
         L     R8,CURWTO               into table message.
         MVC   Ø(LMSG,8),MSG4
         LA    R8,LMSG(R8)             Refresh table message pointer.
         ST    R8,CURWTO
PR123    PR                            POP STACK AND RETURN TO CALLER
         DROP  R8
         EJECT
*
```

```
*
*********************************************************************
* This routine sends the WTOs if one of these is true:              *
* - 2Ø minutes since last sent          or                          *
* - at least one address space is occupying more than 15% of        *
*   auxiliary storage                   or                          *
* - the total auxiliary storage usage is more than 4Ø%.             *
*********************************************************************
*
SEND_MSG DS    ØH
         BAKR  R14,Ø                  PUSH ENVIRONMENT INTO STACK
         CLC   TOTMIN,=F'Ø'
         BE    SENDYES
         CLC   SENDMSG,=C'YES'
         BNE   SENDNO
SENDYES  DS    ØH
         LA    R8,WTOTAB
WTOLOOP  DS    ØH
         MVC   WTOMSG+4(LMSG),Ø(R8)
         WTO   MF=(E,WTOMSG)
         LA    R8,LMSG(R8)
         C     R8,CURWTO
         BL    WTOLOOP
SENDNO   DS    ØH
         L     R8,TOTMIN
         LA    R8,1(R8)
         ST    R8,TOTMIN
         C     R8,MSGTRIG
         BL    PR753951
         LA    R8,Ø
         ST    R8,TOTMIN
PR753951 PR                           POP STACK AND RETURN TO CALLER
         EJECT
*
*
*********************************************************************
* This routine converts a hexadecimal value of pages into its       *
* equivalent in megabytes and edit it.                              *
* At the entry, ADDRESS1 contains the word to translate.            *
* after this routine, ADDRESS2 contains the converted and edited    *
* value.                                                            *
*********************************************************************
*
CONVERT_TO_MEG DS ØH
         BAKR  R14,Ø             PUSH ENVIRONMENT INTO STACK
         SR    R8,R8             CLEAR WORK REGISTER
         L     R9,ADDRESS1       LOAD VALUE TO CONVERT
         SRL   R9,8              PAGE --> MEG
         CVD   R9,PACKWORK       CONVERT INTO DECIMAL
         MVC   ADDRESS2,EDMSKØ8  PREPARE FIELD FOR EDIT
```

```
         ED    ADDRESS2,PACKR    EDIT IT
         PR                      POP STACK AND RETURN TO CALLER
         EJECT
*
*
*********************************************************************
* This routine converts a hexadecimal value of number of slots into  *
* it's % of total number of slots pointed to by MAXSLOTH, and edit it*
* At the entry, ADDRESS1 contains the word to translate.              *
* after this routine, ADDRESS2 contains the converted and edited      *
* value.                                                              *
*********************************************************************
*
CALCUL_PORCENT DS ØH
         BAKR  R14,Ø             PUSH ENVIRONMENT INTO STACK
         LA    R9,MAXSLOTH       A VER MAX NUM OF SLOTS
         CLC   =F'Ø',Ø(R9)       TOTAL NUMBER OF SLOTS = Ø?
         BE    NOAUX             YES (NO AUX STOR FOR EXAMPLE)
         L     R11,ADDRESS1      GET NUMBER OF USED SLOTS
         M     R1Ø,=F'1ØØ'       MULTIPLY BY 1ØØ
         SR    R1Ø,R1Ø           CLEAR R1Ø FOR DIVIDE
         D     R1Ø,Ø(R9)         DIVIDE BY MAX NUMBER OF SLOTS
*                                RESULT IN R11
         CVD   R11,PACKWORK      CONVERT INTO DECIMAL
         MVC   ADDRESS2,EDMSKØ8  PREPARE FIELD FOR EDIT
         ED    ADDRESS2,PACKR    EDIT IT
         PR                      POP STACK AND RETURN TO CALLER
NOAUX    DS    ØH                TO AVOID DIVIDING BY Ø
         MVC   ADDRESS2,=C'        '
         PR                      POP STACK AND RETURN TO CALLER
         EJECT
*
*
*********************************************************************
* This routine checks rc, restores registers, and returns control.
*********************************************************************
*
RETURN   DS    ØH
         LA    R15,Ø
EXIT     RCNTL RC=(15)
         EJECT
*********************************************************************
* Literals for this program.                                        *
*********************************************************************
*
         LTORG
         EJECT
*********************************************************************
* Variables for this program.                                       *
*********************************************************************
```

```
*
         DS    ØD
ADDRESS1 DS    F               USED TO CONVERT TO CHAR
ADDRESS2 DS    D               USED TO CONVERT TO CHAR
MASTASVT DS    F               USED TO STORE MASTER ASCB ASVT addr
ASPNUM   DC    PL3'Ø'          TO CALCUL CURRENT addr SPACE NUMBER
JOBNAME  DS    CL8
MEGTOT   DS    CL8
PCTOCU   DS    CL3
MAXSLOTH DS    F
SLOTVIOH DS    F
SLOTTOTH DS    F
VIOSH    DC    F'Ø'
NVIOSH   DC    F'Ø'
CURWTO   DS    A
TOTMIN   DC    F'Ø'
SENDMSG  DC    C'NO '
*
WTOMSG   WTO   '                                                ',    +
               ROUTCDE=(11),MF=L
MSG1     DC    C'PUAUXMON Total    Aux.    Storage XXXXXXXX Megs.'
LMSG     EQU   *-MSG1
MSG2     DC    C'PUAUXMON Occupied Aux.    Storage XXXXXXXX Megs.'
MSG3     DC    C'PUAUXMON Occupied Aux.    Storage      XXX %.   '
MSG4     DC    C'PUAUXMON AdSpace XXXXXXXX XXX %   XXXXXXXX Megs.'
*
         DS    ØD
PACKWORK DS    PL8             USED FOR CONVERT TO DECIMAL
PACKL    EQU   PACKWORK,4      REDEFINES PACKWORK, LEFT PART
PACKR    EQU   PACKWORK+4,4    REDEFINES PACKWORK, RIGHT PART
*
ASVTà    DS   F
*
*
* Internal table for messages : 1 slot for each address space occupying
* ----------------------------  more than 1% of total auxiliary stor
*                               1ØØ slots maximum
*                               + 1 slot for 1st message
*                               + 2 slots for last two messages
*                               = 1Ø3 slots
*
WTOTAB   DC    1Ø3C'                                             '
*
         EJECT
*********************************************************************
* Constants for this program                                        *
*********************************************************************
*
ONE      DC    PL2'1'
*
```

```
USGTOT   DC    C' 4Ø'       Threshold for % of the total occupied for
*                                 all address spaces.
USGONE   DC    C'  1'       Threshold % occupied for retaining.
USGTWO   DC    C' 15'       Threshold for % of the total occupied for
*                                 one address space.
MSGTRIG  DC    F'2Ø'        How many minutes for next msg, if nothing
*                                 happens.
         DS    ØD
WAITINTV DC    C'ØØØ1ØØØØ'
*                   |---> 1 minute
*
         EJECT
*
*********************************************************************
* Edit masks.                                                       *
*********************************************************************
*
EDMSKØ8  DC    X'4Ø2Ø2Ø2Ø2Ø2Ø212Ø'
         EJECT
*
*********************************************************************
* Flags used for internal logic.                                    *
*********************************************************************
*
#PGMFLAG DC    B'ØØØØØØØØ'
#TSO     EQU   B'1ØØØØØØØ'     TSO address space
#HIGHON  EQU   B'1ØØØØØØØ'     HIGH BIT ON
*
*
*
*********************************************************************
* Dsects                                                            *
*********************************************************************
*
         IHAPSA LIST=YES                      PSA CONTROL BLOCK
         EJECT
         CVT    DSECT=YES,LIST=YES,PREFIX=YES  CVT CONTROL BLOCK
         EJECT
         IHAASVT ,                            ASVT CONTROL BLOCK
         EJECT
         IHAASCB LIST=YES                     addr SPACE CONTROL BLOCK
         EJECT
         IHAASSB LIST=YES                     addr SPACE SECOND BLOCK
         EJECT
         END
```

Michel Joly
Systems Programmer (France)

HOLDDATA research with SMP/E

A difficult aspect of system maintenance is research into system HOLDATA. Often the simple act of finding the appropriate HOLDDATA text is a struggle. SMP/E does a good job of telling you which HOLDDATA you should read, but it does not actually show you the HOLDDATA. Users are forced to hunt for the actual text by browsing the SMPPTS, using the Query dialogue, or even running the LIST command. However, the latest version of SMP/E (Version 3 Release 1) now provides users with the HOLDDATA text by embedding it directly in the SMP/E output. This really saves programmer time.

There are three additional reports produced by SMP/E during APPLY and ACCEPT command processing to show you the HOLDDATA text. These reports contain:

- All of the unresolved HOLDDATA. Unresolved HOLDDATA causes a PTF to fail during APPLY or ACCEPT processing, and typically refers to ERROR HOLDDATA (PEs and Hipers).
- All of the bypassed HOLDDATA. Bypassed HOLDDATA is derived from using the BYPASS operand on the APPLY or ACCEPT command, and typically refers to SYSTEM HOLDDATA.
- A summary of all of the REASON-ids from all of the HOLDDATA that appears in the first two reports.

These reports can become extremely large, and, it can become neccessary to exclude some of the information from the SMP/E output. This is done by specifying the REASON-ids for HOLDDATA to be sup-pressed with a new subentry in the OPTIONS entry called Suppress Holddata (SUPPHOLD).

The resulting SMP/E output from an APPLY or ACCEPT command will provide users with all the HOLDDATA required for system maintenance.

Systems Programmer (UK)

Automating tasks in MVS

The following application is a simple but efficient method to schedule command execution in MVS. They can be console commands or TSO commands.The process works as follows: A started task runs TSO/ batch executing a REXX program. That program is an endless loop, with a specified delay (currently three minutes) between each iteration. The loop allocates and reads a control file and sees if there is any command to execute in the current day and hour. If it does, it executes it. If it does not, it frees the file and ends the loop until the next iteration. This application contains a total of four files, plus a load module for the SLEEP assembler program, which is responsible for creating the delays between iterations.

Those files are:

- AUTOMATE – The started task proc. It should be placed in SYS1.PROCLIB or equivalent. This proc executes IKJEFT01, tht tso/batch program. If there are any REXX or CLISTS that you want to declare for automatic execution, add a SYSEXEC card with the PDS containing them.
- SYSTSIN – Since started tasks cannot contain instream data, it is necessary another file for IKJEFT01 terminal input. It has two TSO commands: a profile noprefix (I always use it under similar circumstances) and the execution of the REXX main program, with the full file name.
- CONTROL – The control file with the commands to execute. I have a comment in the file itself that explains hot to use it. You can specify a single day, a monthly day or a week day with a specific hour. If you need to run a task more than once a day (for example, every hour or so), you must duplicate the lines as needed, each one with its own execution hour. The last columns in the file is where the program registers that the command has already been executed today, by writing today's date there.

In the example file, the first three commands are console commands. They are preceeded by the word MVS. They will be submitted for execution by means of the internal reader declared in the started task.

The first one executes every Wednesday at 15:07, the second only once in a specific date (010915, or September 15, 2001), and the third on the first day of each month. The fourth example is a tso command (submit), and should be executed every day except Saturdays (the seventh column has no 'X'), and twice a day, at 21h and 8h. For this reason, it needs two lines, one for each hour.

AUTOMATE - The REXX program that controls the process. Set the variables at the beginning of it, depending on the location you choose to place the several files. I suggest that you put the control file in a place where it cannot be allocated by any other application, preferibly as a sequential file. This is because the REXX program allocates it as 'old' in every loop iteration and frees it again afterwards, to be able to write execution dates on it, and also to ensure that you are not using it at the time. If the program cannot allocate the control file, it simply leaves the current iteration and goes to sleep waiting for the next, after issuing a message (that you can see at the started task output; the program also issues a message for every executed command). In practise, this means that you are free to edit the control file and make changes to it whenever you want, without the need to bring down the started task.

There are three variables at the beginning of the REXX that control his behaviour: the first is the IPL delay, expressed in seconds. It is a sleeping period that the program observes when it begins to execute, and before entering the main loop. You can set it to zero, if you like. The reason for this is that I have the task starting automatically during a system ipl, but at the same time I do not want it to begin executing things immediately, since it could interfere with the IPL process, or need resources that are not yet available or online.

Seconds is the number of seconds that the program sleepsbetween each loop. You can adjust it freely, according to the frequency of the tasks to execute and the precision of the execution time desired.

Finally, margin is expressed in minutes and is set automatically according to the seconds specified. Currently, margin equals two times seconds: if seconds is 180 (3 minutes), then margin is 6 minutes. This parameter simply represents the upper limit of the execution interval for a given command. For example, if I have a command set

to 18:10, that command will only be executed between that hour and 18:10 plus marginal minutes, that is, 18:16. If, for some reason, the started task is down, or the control file is locked at that time (you might be editing it), the command may not be executed. Currently, margin equal two times the sleep period means that there are two loop iterations that will catch that time span, or two opportunities to execute that command. Why do I do this? Because I do not want to launch a command if the scheduled hour has long gone by and it was not executed then. Once again, the IPL process is a good reason. If I have the system down for some time, when I bring it up again all the commands that were not executed during the down time would be started. In those cases, it is preferable to execute things by hand, if needed.

However, you can disable it, if you want, by enlarging the margin specification: instead of multiplying by two, multiply it by 2000, and all commands will execute, no matter how late they are. But in that case do not forget that if you have multiple daily executions of a single command it could result in multiple attempts to execute it at the same time.

AUTOMATE PROC SOURCE

```
//AUTOMATE PROC
//*
//STEP1    EXEC PGM=IKJEFT01,TIME=1440
//SYSTSIN  DD DISP=SHR,DSN=AUTOMATE.FILE(SYSTSIN)
//INTERDR  DD SYSOUT=(B,INTRDR),DCB=LRECL=80
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
SYSTSIN
```

SYSTSIN SOURCE

```
PROFILE NOPREFIX
EXEC 'AUTOMATE.FILE(AUTOMATE)'
```

AUTOMATE PROGRAM SOURCE

```
/*== REXX MVS ========================================================*/
/*                                                                    */
/* AUTOMATE MVS and TSO commands                                      */
```

```
/*                                                                   */
/*  Components:                                                      */
/*  SYS1.PROCLIB(AUTOMATE)  Started task: program IKJEFT01           */
/*  SYSTSIN   IKJEFT01 input: execute this program                   */
/*  AUTOMATE  This program                                           */
/*  CONTROL   Control file to schedule commands                      */
/*                                                                   */
/*===================================================================*/

automate_control = "AUTOMATE.CONTROL"          /* control file name */
sleepmodule = "My.loadlib(SLEEP)"              /* sleep module      */

delay_ipl = 200     /* Initial delay when this prog starts (seconds) */
seconds = 180       /* Delay between main loop iterations (seconds)  */
margin = seconds / 60 * 2
/*===================================================================*/
/* Column position and length of control file fields                 */
/*===================================================================*/
p1 = 1
p1len = 7                                                 /* Weekdays */
p2 = 9
p2len = 5                                                 /* HH:MM    */
p3 = 15
p3len = 55                                                /* Command  */
p4 = 72
p4len = 8                                                 /* YYYYMMDD */
address tso call "'"sleepmodule"'  '"delay_ipl"'"
do alpha = 0
   hour    = left(time(),5)                               /* HH:MM    */
   today   = date("S")                                    /* YYYYMMDD */
   ymd     = right(today,6)                               /*   YYMMDD */
   day     = right(today,2)                               /*       DD */
   weekday = date("W")                                    /* weekday  */
   select
      when weekday = "Sunday"    then weekday = 1
      when weekday = "Monday"    then weekday = 2
      when weekday = "Tuesday"   then weekday = 3
      when weekday = "Wednesday" then weekday = 4
      when weekday = "Thursday"  then weekday = 5
      when weekday = "Friday"    then weekday = 6
      when weekday = "Saturday"  then weekday = 7
      otherwise nop
   end
   address tso "alloc da('"automate_control"') dd(control) old"
   if rc <> 0 then do
      say rc "Error allocating control file " time() date()
   end
   else do gamma = 1 to 9999
      execute = 0
      execio 1 diskru control
```

```
      if rc <> Ø then leave gamma
      pull linha
      if left(linha,1) = "*" then iterate gamma
      today_file = substr(linha,p4,p4len)
      if today_file = today then iterate gamma
      hour_file = substr(linha,p2,p2len)
      if hour_file > hour then iterate gamma
      else do
         file_minutes = left(hour_file,2)*6Ø + right(hour_file,2)
         minutes = left(hour,2)*6Ø + right(hour,2)
         if minutes - file_minutes > margin then iterate gamma
      end
      if substr(linha,weekday,1) = "X"  then execute = 1
      if word(linha,1) = ymd then execute = 1
      if word(linha,1) = "DAY" &,
         word(linha,2) = day then execute = 1
      if execute = 1 then call execute_command
   end gamma
   execio Ø diskw control "(finis"
   address tso "free dd(control)"
   address tso call "'"sleepmodule"'  '"seconds"'"
end alpha
/*====================================================================*/
/* Subroutines                                                        */
/*====================================================================*/
execute_command:
 comando = strip(substr(linha,p3,p3len))
 say date() time() comando
 if word(comando,1) = "MVS" then call sub_command
 else address tso comando
 linha = overlay(today,linha,p4,p4len)
 queue linha
 execio 1 diskw control
 if rc <> Ø then do
     say "Error writing c_file line" gamma       date() time()
 end
return

sub_command:
 dropbuf
 queue "//AUTOMJB JOB"
 queue "//STEP EXEC PGM=IEFBR14"
 queue "// COMMAND "subword(comando,2)
 "execio * diskw interdr (finis"
return
```

CONTROL FILE EXAMPLE

```
* Control file for AUTOMATE.
*
* Col 1 to 7: Date of execution specification. It can be:
```

```
*  An 'X', meaning a weekday, starting on Sunday(column one).
*  A specific day: "DAY 12 ".
*  A single date, in the format yymmdd: "Ø112Ø3": December 3, 2ØØ1.
*
* Col 9 to 13: Hour (ØØ to 23) and minute (ØØ to 59): 19:15
*
* Col 15 to 7Ø: Command text, TSO command or console command. If it is
*  a console command, put the word "MVS" before.
*
* Col 72 to 79: Last execution date (yyyymmdd). This field is written by
* automate exec after executing a command.
*
* An asterisk in Col 1 means a comment line.
* The line below indicates each field position.
*MTWTFS HH:MM COMMAND.............................................
YYYYMMDD
   X    15:Ø7 MVS 'V 87F,ONLINE'
Ø1Ø915  Ø2:ØØ MVS '$DSPOOL'
DAY Ø1  23:59 MVS 'V NET,ACT,ID=RESC44,SCOPE=ALL'
XXXXXX  21:ØØ SUBMIT 'TRSDE.JCL(POR32)'
XXXXXX  Ø8:ØØ SUBMIT 'TRSDE.JCL(POR32)'
```

SLEEP SOURCE

```
*=====================================================================*
*    SLEEP seconds. Seconds can have one to four digits.              *
*          Without argument, sleep 5 seconds.                         *
*=====================================================================*
SLEEP    AMODE 31
SLEEP    RMODE ANY
SLEEP    CSECT
         STM   R14,R12,12(R13)
         LR    R12,R15
         USING SLEEP,R12
         LR    R15,R13
         LA    R13,SAVEAREA
         ST    R13,8(R15)
         ST    R15,4(R13)
         LR    R5,R1
         L     R2,Ø(Ø,R5)
         LH    R3,Ø(Ø,R2)         R3=parameter length
         LTR   R3,R3
         BZ    PAUSA              No parameter, use default time
         L     R5,2(Ø,R2)         Else, load parm in R5 and shift it
         CH    R3,=H'4'           according to its length
         BE    SHIFTØ
         CH    R3,=H'3'
         BE    SHIFT8
         CH    R3,=H'2'
         BE    SHIFT16
         CH    R3,=H'1'
```

```
         BE    SHIFT24
         B     PAUSA
SHIFT24  SRL   R5,8(Ø)
SHIFT16  SRL   R5,8(Ø)
SHIFT8   SRL   R5,8(Ø)
SHIFTØ   EQU *
         ST    R5,ARGDISP          Convert parm from char to binary
         PACK  ARGPACK,ARGDISP
         CVB   R7,ARGPACK
         MH    R7,=H'1ØØ'          Convert seconds into cents of sec
         ST    R7,SLEEPTIM
*
PAUSA    STIMER WAIT,BINTVL=SLEEPTIM
*
         L     R13,SAVEAREA+4
         LM    R14,R12,12(R13)
         SR    R15,R15
         BR    R14
SAVEAREA DC    18F'Ø'
SLEEPTIM DC    F'5ØØ'              Default sleep in percents of second
ARGDISP  DS    CL4
         DS    ØD
ARGPACK  DS    CL8
         YREGS
         END
```


A REXX program to initialize DASD

For those who work in storage administration, DASD initialization is something that has to be done every once in a while, but, hopefully, a few volumes at a time. However, last time I was faced with the need to initialize some DASD, I was asked for 50 volumes. Not that initializing DASD is hard to do, but getting 50 free addresses, and running through all the steps needed for each one of those volumes is a pain. So, as usually happens when I am faced with a boring task, I wrote a program to do it. The best part, for me at least, is that it will work for 50 volumes, as well as for 1 or 200. In order to be flexible, and easy to use, I decided to use an ISPF panel to obtain the values to work with:


```
COMMAND ===>

                                DISK INIT

        Free volume prefix .......................:     _____
        Prefix for initialization ................:     _____
        Device Type to use .......................:     ____
                DASD Model (if applicable) ........:     _
        Number of Volumes to initialize ..........:     _____
        SMS Volumes  (Y/N) .......................:     _

        JOBNAME to use in JOBs ...................:     ________

                ENTER to Execute
                END   to Cancel
```

In order for this program to be useful at your shop, without any changes, you will need one thing in common with our shop: your available volumes for initialization must be online, and must begin with a common, and standard, prefix. In our case, that prefix is FR: all our free volumes are FR*xxxx*, hence the default used.

This is the way this works: you specify, via the ISPF panel, the DASD prefix to be searched for free volumes, the prefix with which to initialize the new volumes, the device type to use (and model, if you are using 3380/3390), how many volumes to initialize, and whether they are to be SMS managed. I only wrote code to validate 3380 and 3390 device types, but any other kind of DASD can be added without trouble. Becausethis program will submit four JOBs, the JOBNAME to use will have to be specified in here as well. The defaults are assumed prior to the ISPF panel invocation, and, in some cases, are systems dependent, because we have different environments in our different systems. After you specify the values to use, the program will execute an IDCAMS DCOLLECT in order to obtain the volume information to work with. This is done in foreground and, if there are many volumes ONLINE that match the specified prefixes, it may take some time. For this to work, IDCAMS must be defined as an authorized program to run under TSO, in IKJTSOxx AUTHCMD NAMES.

DCOLLECT will produce two output files, one with the free volumes, the other with the DASD that match the new volume prefix. A few

validations will be done, based on the contents of these two files, after which, if everything comes out all right, four JOBs will be submitted, with TYPRUN=HOLD. For this kind of task, I always check each JOB, prior to executing it, via SDSF and a SJ. It is only too easy for something to be painfully wrong.

The first JOB will use an ICEGENER to submit a 'RO *ALL, VARY addr,OFFLINE' for each volume. In a sysplex environment, this will route the VARY command to all the systems. If you are not in a sysplex environment, take out the 'RO *ALL',. However, you will have to devise a way to do the VARY ONLINE on the other systems that share the device. The third one will do the ONLINE. The submitted JCL will look like this, and the third JCL will differ only in the OFFLINE keyword, and in the description;

```
//jobname JOB (ACCT#),'PUT VOLUMES OFFLINE',
//            MSGLEVEL=(1,1),
//            TYPRUN=HOLD,
//            CLASS=W,MSGCLASS=X
//*
//PUT#OFF  EXEC PGM=ICEGENER
//SYSPRINT DD  SYSOUT=*
//SYSUT2   DD  SYSOUT=(*,INTRDR)
//SYSIN    DD  DUMMY
//SYSUT1   DD  DATA,DLM='ff'
/*$VS,'RO *ALL,V xxxx,OFFLINE'
ff
/*
```

The second JOB will run ICKDSF and the INITs for all the volumes, with VERIFY(VOLSER) and, if the volumes are to be SMS managed, with the STORAGEGROUP keyword as well. The VTOC and INDEXED VTOC size are device type and model dependent, and hard coded in the program, within a SELECT, so it will be easy to accommodate new/old types. The submitted JCL will look like this:

```
//jobname JOB (ACCT#),'INIT VOLUMES OFFLINE',
//            MSGLEVEL=(1,1),
//            TYPRUN=HOLD,
//            CLASS=W,MSGCLASS=X
//*
//INIT#OFF EXEC PGM=ICKDSF,PARM='NOREPLYU'
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
   INIT UNIT(xxxx) INDEX(0,1,014) VTOC(1,0,060) -
    VERIFY(FRxxxx) VOLID(prf01) STORAGEGROUP
/*
//*
```

The fourth JOB will be generated only if the volume is to be SMS managed and it will create the VVDS. If you want the VVDS to be created, even for non-SMS volumes, you will only have to remove an IF in the GENERATE_JOBS procedure. This submitted JCL will look like this:

```
//jobname JOB (ACCT#),'CREATE VVDS',
//            MSGLEVEL=(1,1),
//            TYPRUN=HOLD,
//            CLASS=W,MSGCLASS=X
//*
//CRE#VVDS EXEC PGM=IDCAMS,REGION=4M
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
  DEFINE CLUSTER (NAME(SYS1.VVDS.Vprfnnn)    -
    VOL(prfnnn) NONINDEXED TRK(45 Ø))
/*
```

Some validations are made at panel level:

- Volume prefixes length (must be at least one character).
- Device type and model (3380/J/K 3390/1/2/3. Others can be easily added).
- Number of volumes to initialize (must be numeric and greater than zero).
- SMS managed (must be Y or N).
- JOBNAME length (must be 8).

This is the DISKINIT panel definition:

```
)ATTR
  # TYPE(INPUT)  INTENS(LOW)  COLOR(TURQ)   CAPS(ON) PADC('_')
  | TYPE(TEXT)   INTENS(LOW)  COLOR(BLUE)   SKIP(ON)
  @ TYPE(TEXT)   INTENS(HIGH) COLOR(WHITE)  SKIP(ON)
  + TYPE(TEXT)   INTENS(HIGH) COLOR(YELLOW) SKIP(ON)
  £ TYPE(TEXT)   INTENS(LOW)  COLOR(TURQ)   SKIP(ON)
)BODY
|-----------------------------------------------
@COMMAND ===>#ZCMD
|
|
+                                DISK INIT
|
|         Free volume prefix .....................:    #Z    |
|         Prefix for initialization ..............:    #Z    |
|         Device type to use .....................:    #Z    |
```

```
|                 DASD Model (id applicable) ........:    #Z    |
|         Number of Volumes to initialize ...........:    #Z    |
|         SMS Volumes £(Y/N)|.......................:    #Z    |
|
|         JOBNAME to use in JOBs ...................:    #Z    |
|
|                @ENTER|to Execute
|                @END  |to Cancel
|
)INIT
   .ZVARS='(VOLPREF NEWPREF TYPE MODEL HOWMANY SMS JOBNAME)'
)PROC
   VER (&VOLPREF,LEN,GE,1)
   VER (&NEWPREF,LEN,GE,1)
   VER (&TYPE,NONBLANK,LIST,3380,3390)
   IF  (VER(&TYPE,LIST,3380))
       VER(&MODEL,NONBLANK,LIST,J,K)
   ELSE
       IF  (VER(&TYPE,LIST,3390))
           VER(&MODEL,NONBLANK,LIST,1,2,3)
       ELSE
           &MODEL = ' '
   VER (&HOWMANY,NONBLANK,NUM)
   VER (&SMS,NONBLANK,LIST,S,N)
   VER (&JOBNAME,LEN,EQ,8)
   &PFKEY = .PFKEY
)END
```

In order to obtain free gaps in the volume numeration, the program will order the second output file, generated by DCOLLECT, by VOLSER, using an EDIT macro (X$INDK$X) in the beginning of the GET_NEW_VOL procedure. It is a very simple macro:

```
/* REXX
-                                                                    */
address "ISREDIT"
"macro"
"sort 25 30"
"save"
"end"
return
/* - - - - - - - - - - - - - - */
```

If there are enough volumes to satisfy your request, the INIT related jobs will be generated and submitted, and a LOG record will be formatted for each DASD volume initialized. These records will be added to a LOG file (which will be created in the first use of this program), and will be recorded in a temporary file, which will be browsed, prior to program termination, so you will have visual

information of the volumes selected:

```
BROWSE    userid.D01241.T121231.VIEW.LOG           Line 000000000 Col 001 080
 Command ===>                                          Scroll ===> CSR
***************************** Top of Data ******************************
FRxxxx init as newprf on yyyymmdd - hh:mm:ss by RACF user name
***************************** Bottom of Data ***************************
```

The program is as follows:

```
/* REXX
-                                                                    */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Set Default values                                                */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
howmany=1
volpref="FR"                /* Free Volume PREFIX                   */
newpref="CPG"               /* New Volume PREFIX                    */
devtype=3390                /* Default Device Type                  */
model=3                     /* Default type model                   */
sms="Y"                     /* SMS init is the default              */
jobname=userid()"I"         /* JOB NAME to use                      */
acctnum="ACCT#"             /* ACCOUNT info to use                  */
job_class="A"               /* JOB CLASS to use                     */
msg_class="X"               /* MSG CLASS to use                     */
log_hlq="PDOIDM"            /* LOG FILE High Level Qualifier        */
zedlmsg=""
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Invoke ISPF Panel to specify invocation values                   */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
do  z=1
    address "ISPEXEC" "control errors return"
    address "ISPEXEC" "display panel(diskinit)"
    select
        when wordpos(pfkey,"PF03 PF15 PF04 PF16")>0 then
            do
                zedlmsg=zedlmsg,
                    "You terminated the DISKINIT process, by",
                    "pressing the PF03/15 or PF04/16 Key"
                leave z
            end
        otherwise
            l=6-length(newpref)
            limit=copies(9,l)
            if  limit<howmany then
                do
                    zedlmsg=zedlmsg,
                        "You specified more volumes to initialize",
                        "("howmany") than it is possible with prefix",
                        newpref" ("limit")"
                end
```

```
            else
                do
                    call alloc_files
                    leave z
                end
    end
    if  zedlmsg¬=""  then
        do
            address "ISPEXEC" "SETMSG MSG(ISRZØØ1)"
            zedlmsg=""
        end
end
if  zedlmsg ¬= "" then
    do
        address "ISPEXEC" "SETMSG MSG(ISRZØØ1)"
    end
return
/* - - - - - - - - - - - - - */
alloc_files:
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* ALLOCATE Work files                                                    */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
"alloc f(sysprint) shr reuse dummy"
hlqs=userid(),              /* high level qualifiers for work files  */
    ||".D"date("J"),
    ||".T"space(translate(time(),,":"),Ø)
out_dsn_1="'"hlqs".OUTFILE.#Ø1'"   /* Work File for Free Volumes     */
out_dsn_2="'"hlqs".OUTFILE.#Ø2'"   /* Work file for New Volumes      */
dd#1="A"time("S")
dd#2="B"time("S")
"alloc f("dd#1") new dsorg(PS) recfm(V B) lrecl(454)",
    "da("out_dsn_1") space (1Ø 5) tracks release"
if  rc=Ø then
    do
        "alloc f("dd#2") new dsorg(PS) recfm(V B) lrecl(454)",
            "da("out_dsn_2") space (1Ø 5) tracks release"
        if  rc=Ø then
            do
                in_dsn="'"hlqs".SYSIN'"
                "alloc f(SYSIN)  new dsorg(PS) recfm(F B) lrecl(8Ø)",
                    "da("in_dsn") space (1 1) tracks release reuse"
                if  rc=Ø then
                    do
                        call format_sysin
                        "alloc f(sysin)    shr reuse da(*)"
                    end
                else
                    do
                        zedlmsg=zedlmsg,
                            "Error ("rc") on the ALLOC for "in_dsn
                        "free f("dd#1","dd#2")"
                    end
```

```
            end
        else
            do
                zedlmsg=zedlmsg,
                    "Error ("rc") on the ALLOC for "out_dsn_2
                "free f("dd#1")"
            end
    end
else
    do
        zedlmsg=zedlmsg,
            "Error ("rc") on the ALLOC for "out_dsn_1
    end
"alloc f(sysprint) shr reuse da(*)"
return
/* - - - - - - - - - - - - - */
format_sysin:
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Format DCOLLECT SYSIN and call IDCAMS                                  */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
queue"  DCOLLECT OFILE("dd#1") VOL("volpref"*) NODATAINFO"
queue"  DCOLLECT OFILE("dd#2") VOL("newpref"*) NODATAINFO"
"execio "queued()" diskw SYSIN (finis)"
if  rc=0 then
    do
        "alloc f(sysin) reuse old da("in_dsn") delete"
        "CALL *(IDCAMS)"
        if  rc=0 then
            do
                "alloc f("dd#1") old da("out_dsn_1") delete"
                "execio * diskr "dd#1" (finis stem volume_info.)"
                if  rc=0 then
                    do
                        if  volume_info.0>0 then
                            do
                                call process_data
                            end
                        else
                            do
                                zedlmsg=zedlmsg,
                                    "No volumes "volpref,
                                    "* were obtained"
                            end
                    end
                else
                    do
                        zedlmsg=zedlmsg,
                            "Error ("rc") on "out_dsn_1" READ"
                    end
                "free f("dd#1")"
            end
        else
```

```
            do
                zedlmsg=zedlmsg,
                    "Error ("rc") during DCOLLECT execution.",
                    "Process state unknown."
            end
    end
else
    do
        zedlmsg=zedlmsg,
            "Error ("rc") on the WRITE for "in_dsn
        "dropbuf"
    end
return
/* - - - - - - - - - - - - - - */
process_data:
if  volume_info.0<howmany then
    do
        zedlmsg=zedlmsg,
            "Not enough volumes "volpref"* to initialize",
            "You asked for "howmany", but only "volume_info.0,
            "are available"
    end
else
    do
        /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
        /* Set VTOC, VTOCIX an VVDS sizes, depending on DASD device  */
        /*                                 and model type            */
        /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
        select
            when devtype=3390 then
                do
                    if  model=3 then
                        do
                            vtoc="1,0,060"
                            vtix="0,1,014"
                            vvds="45 0"
                        end
                    else
                        do
                            vtoc="1,0,045"
                            vtix="0,1,014"
                            vvds="30 0"
                        end
                end
            when devtype=3380 then
                do
                    if  model="K" then
                        do
                            vtoc="1,0,060"
                            vtix="0,1,014"
                            vvds="45 0"
                        end
```

```
            else
                do
                    vtoc="1,Ø,Ø45"
                    vtix="Ø,1,Ø14"
                    vvds="3Ø Ø"
                end
        end
    otherwise
        nop
end
o=Ø
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Process DCOLLECT obtained information for free volumes          */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
do  a=1 to volume_info.Ø
    parse value volume_info.a with 25 volid,
                                   31 .,
                                   45 tot_cap,
                                   49 .,
                                   69 dev_type,
                                   77 dev_num,
                                   79 .
    dev_type=strip(dev_type)
    dev_num =c2x(dev_num)
    tot_cap =c2d(tot_cap)
    /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
    /* Set Model Type, based on DASD total capacity                   */
    /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
    select
        when dev_type="339Ø" then
            do
                select
                    when tot_cap>2771ØØ then
                        model_a="9"
                    when tot_cap>1847ØØ then
                        model_a="3"
                    when tot_cap> 923ØØ then
                        model_a="2"
                    otherwise
                        model_a="1"
                end
            end
        when dev_type="338Ø" then
            do
                select
                    when tot_cap>123Ø9ØØ then
                        model_a="K"
                    when tot_cap> 6154ØØ then
                        model_a="E"
                    otherwise
                        model_a="J"
                end
```

```
                    end
                otherwise
                    model_a=""
            end
            /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
            /* Is this DASD of the same type & model as specified?   */
            /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
            if  devtype=dev_type & model=model_a then
                do
                    o=o+1
                    addr.o  =dev_num
                    volid.o =volid
                    if  o=how_many then
                        leave a
                end
        end
        /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
        /* Do we have as many free DASD as requested?                    */
        /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
        select
            when o=Ø then
                do
                    zedlmsg=zedlmsg,
                        "You asked for "howmany devtype"/"model,
                        "volumes, but there are none available"
                end
            when o<howmany then
                do
                    zedlmsg=zedlmsg,
                        "You asked for "howmany devtype"/"model,
                        "volumes, but there are only "o" available"
                end
            otherwise
                call get_new_vol
        end
    end
return
/* - - - - - - - - - - - - - - */
get_new_vol:
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Process DCOLLECT obtained information for new volumes                 */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
drop volume_info.
        /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
        /* Edit work file and order by volser                            */
        /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
address "ISPEXEC" "edit dataset("out_dsn_2") macro(x$indk$x)"
"alloc f("dd#2") old da("out_dsn_2") delete"
"execio * diskr "dd#2" (finis stem volume_info.)"
if  rc=Ø then
    do
        first=Ø
```

```
        /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
        /* Are there any volumes with the specified prefix?              */
        /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
        if  volume_info.Ø>Ø then
            do  a=1 to volume_info.Ø
                parse value volume_info.a with 25 nnn 31 .
                nnn=right(nnn,1)
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                /* Is the suffix a valid whole number ?                  */
                /* - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                if  datatype(nnn,"W") then
                    do
                        if  nnn-first>=howmany+1 then
                            do
                                leave a
                            end
                        else
                            do
                                first=nnn
                            end
                    end
                else
                    nop
            end
        /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
        /* Is the available range big enough ?                           */
        /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
        if  first+howmany>limit then
            do
                zedlmsg=zedlmsg,
                    "The number of volumes to initialize ("howmany")",
                    "exceeds the available range : "first+1" to "limit
            end
        else
            do
                call generate_jobs
            end
    end
else
    do
        zedlmsg=zedlmsg,
            "Error ("rc") on "out_dsn_2" READ"
    end
"free f("dd#2")"
return
/* - - - - - - - - - - - - - - */
generate_jobs:
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Format, and SUBMIT, the INIT process related JOBs                     */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
name=name()                  /* get RACF user name                       */
do  a=1 to howmany
```

```
    new_value=first+a
    new_vol.a=newpref||right(new_value,1,"Ø")
    log_line.a=volid.a" init as "new_vol.a,
        "on "date("S")" - "time()" by "name
end
call put_offline             /* format PUT OFFLINE JOB               */
call init_offline            /* format INIT OFFLINE JOB              */
call put_online              /* format PUT ONLINE JOB                */
ddname="O"time("S")
"alloc f("ddname") writer(intrdr) sysout(A) LRECL(8Ø) RECFM(F)"
"execio "off_line.Ø" diskw "ddname" (stem off_line.)"
"execio "init_off.Ø" diskw "ddname" (stem init_off.)"
"execio "on_line.Ø"  diskw "ddname" (stem on_line.)"
if  sms="Y" then
    do
        job#=4
        call create_vvds     /* format CREATE VVDS JOB               */
        "execio "cr_vvds.Ø" diskw "ddname" (stem cr_vvds.)"
    end
else
    do
        job#=3
    end
"execio Ø  diskw "ddname" (finis)"
"free f("ddname")"
if  howmany=1 then
    do
        zedlmsg=zedlmsg,
            job#" Jobs ("jobname"), to Initialize "new_vol.1,
            "have been submitted"
    end
else
    do
        zedlmsg=zedlmsg,
            job#" Jobs ("jobname"), to Initialize "new_vol.1" to",
            new_vol.howmany", have been submitted"
    end
call log_down
return
/* - - - - - - - - - - - - - - */
put_offline:
/* Format the VARY OFFLINE JCL                                       */
queue"//"jobname" JOB ("acctnum"),'PUT VOLUMES OFFLINE',"
queue"//            MSGLEVEL=(1,1),"
queue"//            TYPRUN=HOLD,NOTIFY=&SYSUID,"
queue"//            CLASS="job_class",MSGCLASS="msg_class
queue"//*"
queue"//PUT#OFF  EXEC PGM=ICEGENER"
queue"//SYSPRINT DD  SYSOUT=*"
queue"//SYSUT2   DD  SYSOUT=(*,INTRDR)"
queue"//SYSIN    DD  DUMMY"
queue"//SYSUT1   DD  DATA,DLM='££'"
```

```
do  a=1 to howmany
    queue"/*$VS,'RO *ALL,V "addr.a",OFFLINE'"
end
queue"ff"
queue"/*"
do  a=1 to queued()
    pull off_line.a
end
off_line.0=a-1
return
/* - - - - - - - - - - - - - */
init_offline:
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Format the INIT OFFLINE JCL                                             */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
queue"//"jobname" JOB ("acctnum"),'INIT VOLUMES OFFLINE',"
queue"//            MSGLEVEL=(1,1),"
queue"//            TYPRUN=HOLD,NOTIFY=&SYSUID,"
queue"//            CLASS="job_class",MSGCLASS="msg_class
queue"//*"
queue"//INIT#OFF EXEC PGM=ICKDSF,PARM='NOREPLYU'"
queue"//SYSPRINT DD SYSOUT=*"
queue"//SYSIN    DD *"
if  sms="Y" then
    opts="STORAGEGROUP"
else
    opts=""
do  a=1 to howmany
    queue"   INIT UNIT("addr.a") INDEX("vtix") VTOC("vtoc") -"
    queue"    VERIFY("volid.a") VOLID("new_vol.a") "opts
end
queue"/*"
queue"//*"
do  a=1 to queued()
    pull init_off.a
end
init_off.0=a-1
return
/* - - - - - - - - - - - - - */
put_online:
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Format the VARY ONLINE JCL                                              */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
queue"//"jobname" JOB ("acctnum"),'PUT VOLUMES ONLINE',"
queue"//            MSGLEVEL=(1,1),"
queue"//            TYPRUN=HOLD,NOTIFY=&SYSUID,"
queue"//            CLASS="job_class",MSGCLASS="msg_class
queue"//*"
queue"//PUT#ON   EXEC PGM=ICEGENER"
queue"//SYSPRINT DD  SYSOUT=*"
queue"//SYSUT2   DD  SYSOUT=(*,INTRDR)"
queue"//SYSIN    DD  DUMMY"
```

```
queue"//SYSUT1   DD  DATA,DLM='ff'"
do  a=1 to howmany
    queue"/*$VS,'RO *ALL,V "addr.a",ONLINE'"
end
queue"ff"
queue"/*"
do  a=1 to queued()
    pull on_line.a
end
on_line.0=a-1
return
/* - - - - - - - - - - - - - - */
create_vvds:
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Format the CREATE VVDS JCL                                                */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
queue"//"jobname" JOB ("acctnum"),'CREATE VVDS',"
queue"//            MSGLEVEL=(1,1),"
queue"//            TYPRUN=HOLD,NOTIFY=&SYSUID,"
queue"//            CLASS="job_class",MSGCLASS="msg_class
queue"//*"
queue"//CRE#VVDS EXEC PGM=IDCAMS,REGION=4M"
queue"//SYSPRINT DD  SYSOUT=*"
queue"//SYSIN    DD  *"
do  a=1 to howmany
    queue"  DEFINE CLUSTER (NAME(SYS1.VVDS.V"new_vol.a")    -"
    queue"    VOL("new_vol.a") NONINDEXED TRK("vvds"))"
end
queue"/*"
do  a=1 to queued()
    pull cr_vvds.a
end
cr_vvds.0=a-1
return
/* - - - - - - - - - - - - - - */
name: procedure
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Get user name from RACF profile                                           */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
x=outtrap(user.,,"NOCONCAT")
"LU "userid()
x=outtrap("OFF")
parse value user.1 with "NAME="who"OWNER="
return strip(name)
/* - - - - - - - - - - - - - - */
log_down:
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Write the INIT information on the LOG file                                */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
dd="O"time("S")
log_dsn="'"log_hlq".##LOG.INITDASD'"
sysmsg=sysdsn(log_dsn)
```

```
if  sysmsg="OK" then
    do
        "alloc f("dd") mod reuse da("log_dsn")"
    end
else
    do
        "alloc f("dd") new reuse da("log_dsn")",
            "lrecl (254) recfm(V B) space (1Ø 5) tracks"
    end
if  rc¬=Ø then
    do
        zedlmsg=zedlmsg,
            "Error ("rc") on the ALLOC for "log_dsn
    end
else
    do
        "execio "howmany" diskw "dd" (finis stem log_line.)"
        if  rc¬=Ø then
            do
                zedlmsg=zedlmsg,
                    "Error ("rc") on the WRITE for "log_dsn
            end
        "Free f("dd")"
    end
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* View this execution LOG, using a temporary work file                 */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
dd="W"time("S")
view_log="'"hlqs".VIEW.LOG'"
"alloc f("dd") new reuse da("view_log")",
    "lrecl (254) recfm(V B) space (1Ø 5) tracks delete"
if  rc\=Ø then
    do  a=1 to how_many
        say log_line.a
    end
else
    do
        "execio "howmany" diskw "dd" (finis stem log_line.)"
        if  rc=Ø then
            do
                address "ISPEXEC" "control errors return"
                address "ISPEXEC" "browse dataset("view_log")"
            end
        else
            do  a=1 to how_many
                say log_line.a
            end
        "Free f("dd")"
    end
return
/* - - - - - - - - - - - - - - */
```

The Integrated Facility for Linux for the Multiprise 3000

In October 2001 IBM released the Integrated Facility for Linux (IFL) on the Multiprise 3000, for running single or multiple Linux images. This optional facility was previously available only on the zSeries and System/390 G5 and G6 models. It enables Linux to be run natively as a stand-alone or as a logical partition on a zSeries 900 and S/390. Linux applications can be isolated on the Multiprise 3000 in their own workspace, to provide dedicated Linux workload capacity. These 'Linux' MIPS are not counted for the System/390 software running on the first engine.

The standard processor configuration of the model H30 with IFL feature is a uniprocessor running at H30 speeds, a uniprocessor configured as the IBM Integrated Facility for Linux and running at H50 levels, one System Assist Processor (SAP), and 4GB real memory available for all LPARs. The standard processor configuration of the model H50 with IFL feature is a uni-processor running at H50 levels, a uni-processor configured as the IFL and running at H50 speeds, one SAP, and 4GB real memory available for all LPARs. IFL is also available as an upgrade to existing H30 and H50 systems.

The System/390 Virtual Image Facility for Linux and z/VM Version 4 enable the running of more Linux images than can be deployed using LPARs, and provide capabilities to help create and manage these images. If these are used in conjunction with the IFL it could provide a viable means of consolidating servers on the Multiprise. Even a small box like the Multiprise can act as a central point to consolidate a large number of departmental servers such as Web servers, e-mail servers and file/print servers. With the considerable increases in costs many users are seeing with Microsoft's new licensing initiatives this could be an ideal means of saving money by moving servers from expensive Windows NT and 2000 boxes to a Multiprise with IFL. The key issue is for IT staff to audit their departmental servers and see how many can be consolidated and what cost benefits this will have. This is also a useful way of acquiring Linux skills, which will be a useful asset when Linux eventually begins to make inroads onto the desktop.

Lost ASVT entries

A while ago, I found a critical situation where MVS was running out of available slots in the ASVT. I did not understand because I sized this table at least five times the number of expected address spaces in my peak hours. Asking IBM for support, they gave me this interesting answer:

"The most common reason for a slot in the ASVT to be marked non-reusable is that the corresponding address space has gone down with cross-memory connections or binds. Due to system integrity concerns, the asid cannot be reused until these binds are broken. Please see APAR OY26621 for additional information about this problem of cross-memory binds and what operational procedures may be performed to minimize the number of address spaces that terminate with unbreakable cross-memory connections. Note that the termination of DB2 regions is a major cause of slots being marked non-reusable. At OS/390 Release 1 Version 2 and earlier, the system provides no way of identifying the job last associated with an address space that has been marked non-reusable. From OS/390 Version 1 Release 3 and later, the XMSE control block in the private area of the PCAUTH address space (asid X'0002') contains the last job name assigned to its corresponding address space. II08563 details how to locate the XMSE from the ASCB or ASSB."

So I wrote this batch program to help me in determining which address spaces are causing me trouble.

For the records, I found RMF guilty, because, for some obscure reason, production people decide to stop it three times every day, leaving two lost ASVT slots at each stop – one for RMF itself, one for RMFGAT.

Note: this program uses two 'in-house' macros:

- INITL – to start the program (get some memory for save area, chaining of save areas, register equates).
- RCNTL – at the end of the program (restore registers, free of save area, and return).

You can substitute these with your own.

```
PUXMSE CSECT
PUXMSE AMODE   31
PUXMSE RMODE   24
*************************************************************
* This program is written to help in determine why we are    *
* losing slots in the ASVT.                                  *
*************************************************************
* Environment:                                               *
* This program should work from OS/390 1.3 and up.           *
* It was fully tested under OS/390 2.4 and 2.5               *
*************************************************************
* Warning:  Part of this program goes into PCAUTH's private  *
*           area and retrieves some information using CROSS- *
*           MEMORY. So it should be link-edited with AC(1)   *
*           and loaded from an authorized library.           *
*************************************************************
* Main logic:                                                *
*                                                            *
* CVT ---> ASVT ---> ASCB of PCAUTH ---> ASSB of PCAUTH      *
* ASSB of PCAUTH ---> XMSE of PCAUTH                         *
* XMSE of PCAUTH ---> XMSE next,  etc                        *
* XMSE ---> SETC                                             *
*                                                            *
* - Are we authorized (APF) ?              VERIF_AUTH        *
* - Open output file LISTXMSE              OPENDCBS          *
* - Search the ASVT for PCAUTH             SEARCH_PCAUTH     *
* - Write 1st title line on output         WRITE_LINE1       *
* - Write 2nd title line on output         WRITE_LINE2       *
* - Process PCAUTH's private area          PROCESA_XMSE      *
*   - Get an ALET for cross-memory work    ALESERV_ADD       *
*   - Extract info from PCAUTH             CROSS_MEM         *
*   - Free the ALET                        ALESERV-DEL       *
* - Write last lines of total              WRITE_TOTAL       *
* - Close output file                      CLOSDCBS          *
* - Return to MVS                          RETURN            *
*************************************************************
* INPUT  : - Nothing                                         *
* OUTPUT : - The DD LISTXMSE (FBA lrecl 133) contains        *
*            the detail of the XMSE blocks.                  *
*************************************************************
* JCL to execute this program  :                             *
*                                                            *
*  //XMSEINFO  EXEC PGM=PUXMSE                               *
*  //STEPLIB   DD   DISP=SHR,DSN=my.load                     *
*  //SYSUDUMP  DD   SYSOUT=*                                 *
*  //LISTXMSE  DD   SYSOUT=*                                 *
*************************************************************
* Lked attributs :                                           *
```

```
* Amode 31                                                      *
* Rmode 24                                                      *
* AC     1                                                      *
****************************************************************
* This program will return you the following information      *
* in the LISTXMSE dd :                                          *
* - on first line with                                          *
*      . PCAUTH ASCB address                                    *
*      . PCAUTH ASSB address                                    *
*      . PCAUTH XMSE address                                    *
* - one line for each XMSE block with                           *
*      . XMSE address                                           *
*      . Job name                                               *
*      . Asid number                                            *
*      . XMSE previous address                                  *
*      . XMSE next     address                                  *
*      . ASCB address                                           *
*      . '** lost  **' if the block is for a lost entry         *
*      . SETC address                                           *
*      . SysLX         in case of using it                      *
*      . Number of to/from cross-memory connections for this    *
*              address space                                    *
* - one line with                                               *
*      . Number of slots in the ASVT                            *
* - one line with                                               *
*      . Max users from Parmlib (IEASYSxx)                      *
* - one line with                                               *
*      . Number of lost entries in the ASVT                     *
****************************************************************
         EJECT
*********************************************************************
* Return codes :                                                    *
*                                                                   *
* Ø  : OK                                                           *
* 4  :                                                              *
* 8  : Problem in scanning the XMSE chain in PCAUTH's EPVT.         *
* 12 : Problem to obtain an ALET (cross-mem)                        *
* 16 : We didn't find PCAUTH                                        *
* 2Ø : Error opening LISTXMSE out file                              *
* 24 : Program not authorized                                       *
*********************************************************************
* Messages :                                                        *
*                                                                   *
* - PUXMSEØ1 program not authorized (APF).                          *
*   This program must be loaded from an authorized library and      *
*   link-edited with AC(1).                                         *
*   The program stops with RC=24.                                   *
*   See VERIF_AUTH routine.                                         *
*                                                                   *
* - PUXMSEØ2 error opening LISTXMSE out file.                       *
```

```
*   See the LISTXMSE dd in your JCL.                                *
*   The program stops with RC=2Ø.                                   *
*   See OPENDCBS routine.                                           *
*                                                                   *
* - PUXMSEØ3 PCAUTH not found.                                      *
*   Scanning the ASVT, we didn't find the slot corresponding        *
*   to the PCAUTH address space (normally it's the second).         *
*   The program stops with RC=16.                                   *
*   See SEARCH_PCAUTH routine.                                      *
*                                                                   *
* - PUXMSEØ4  unable to obtain ALET                                 *
*             XXXXXXXX is the return code                           *
*   We was not able to get an ALET for connection with PCAUTH       *
*   address space.                                                  *
*   The program stops with RC=12.                                   *
*   See ALESERV_ADD routine.                                        *
*                                                                   *
* - PUXMSEØ5  Problem with XMSE chain into                          *
*   PUXMSEØ5  the PCAUTH EPVT.                                      *
*   Scanning the XMSE chain into the PCAUTH private area, we didn't *
*   find the acronym for one of those blocks.                       *
*   The program stops with RC=Ø8.                                   *
*   See CROSS_MEM routine.                                          *
*********************************************************************
* Conventions:                                                      *
*                                                                   *
* $ Prefixed fields are counters                                    *
* £ Prefixed fields are part of output lines                        *
* # Prefixed fields are flags                                       *
*********************************************************************
* Register usage:                                                   *
*                                                                   *
* RØ  : reserved                                                    *
* R1  : reserved for macros                                         *
* R2  : reserved for trt instruction                                *
* R3  : first base register                                         *
* R4  : not used                                                    *
* R5  : not used                                                    *
* R6  : not used                                                    *
* R7  : not used                                                    *
* R8  : work register                                               *
* R9  : work register                                               *
* R1Ø : work register                                               *
* R11 : work register                                               *
* R12 : work register                                               *
* R13 : reserved as savearea pointer                                *
* R14 : reserved as link register (return address)                  *
* R15 : reserved for return code                                    *
*********************************************************************
         EJECT
```

```
*********************************************************************
* Some housekeeping. R3, base register.                             *
*********************************************************************
         INITL 3,EQU=R
         EJECT
*********************************************************************
* Main logic                                                        *
*********************************************************************
         BAS   R14,VERIF_AUTH      Authorized?
         BAS   R14,OPENDCBS        Open OUTPUT file
         TM    #PGMFLAG,#NOTAUTH   Flag authorized ?
         BO    RETURN              No, terminate processing rc=24
         TM    #PGMFLAG,#OPENERR   Open error?

         BO    RETURN              Yes, terminate processing rc=20
         BAS   R14,SEARCH_PCAUTH   Search for PCAUTH address space
         TM    #PGMFLAG,#PCANOTF   Found it?
         BO    CLOSE               No, terminate processing rc=16
         BAS   R14,WRITE_LINE1     1st title line on LISTXMSE
         BAS   R14,WRITE_LINE2     2nd title line on LISTXMSE
         BAS   R14,PROCESA_XMSE    Let's do the real work ...
         BAS   R14,WRITE_TOTAL     last lines of total on LISTXMSE
CLOSE    BAS   R14,CLOSDCBS        Close all DCBs
         B     RETURN              Bye
         EJECT
*********************************************************************
* This routine checks if we are APF authorized.                     *
*********************************************************************
VERIF_AUTH DS  0H
         BAKR  R14,0                    PUSH ENVIRONMENT INTO STACK
         TESTAUTH FCTN=1                LET SEE IF WE ARE AUTHORIZED
         LTR   15,15                    IF YES,
         BZ    PR10008                  RETURN
         OI    #PGMFLAG,#NOTAUTH        IF NOT, INDICATE SO
         WTO   'PUXMSE01 program not authorized (APF).    ',ROUTCDE=11
PR10008  DS    0H
         PR                             POP STACK AND RETURN TO CALLER
         EJECT
*********************************************************************
* This routine open all DCBs that we need in this program           *
*********************************************************************
OPENDCBS DS    0H
         BAKR  R14,0               PUSH ENVIRONMENT INTO STACK
         USING IHADCB,R11          BASE FOR DCB DSECT
         OPEN  (LISTXMSE,OUTPUT)
         LA    R11,LISTXMSE        R11 = DCB #
         TM    DCBOFLGS,X'10'      GOOD OPEN?
         BO    OPEN_OK             YES, GO TO PROCESS
         WTO   'PUXMSE02 error opening LISTXMSE out file.',ROUTCDE=11
         OI    #PGMFLAG,#OPENERR   SET OPEN_ERROR FLAG
```

```
         B     PRØØ1Ø              RETURN TO CALLER
OPEN_OK  DS    ØH
         DROP  R11                 FREE R11
PRØØ1Ø   PR                        POP STACK AND RETURN TO CALLER
         EJECT
*********************************************************************
* This routine search the ASVT for the PCAUTH's ASCB.               *
* From there, we get the ASSB and XMSE adresses.                    *
*********************************************************************
SEARCH_PCAUTH  DS ØH
         BAKR  R14,Ø                  PUSH ENVIRONMENT INTO STACK
         L     R9,16                  GET CVT ADDRESS
         USING CVT,R9                 ESTABLISH ADDRESSABILITY
         L     R9,CVTASVT             GET ASVT ADDRESS
         USING ASVT,R9                ESTABLISH ADDRESSABILITY
         MVC   MAXI,ASVTMAXI          Save max users
         L     R12,ASVTMAXU           GET MAX NUMB # SPACE FOR LOOP
         ST    R12,MAXU               Save for future use
         LA    R11,ASVTENTY           GET # OF FIRST ENTRY
         ST    R11,MASTASVT           SAVE MASTER ASVT ENTRY #
         OI    MASTASVT,#HIGHON       HIGH BIT ON
ASCBLOOP DS    ØH
         TM    Ø(R11),ASVTRSAV        VALID ASCB ?
         BO    RUNLOOP                NO, CHECK NEXT ASVT ENTRY
         L     R8,Ø(R11)              GET ASCB #
         USING ASCB,R8                ESTABLISH ADDRESSABILITY
         L     R2,ASCBJBNI            GET # INITIATED JOBNAME
         CLC   Ø(8,R2),=C'PCAUTH  '   IS IT OUR ADDRESS SPACE ?
         BE    BINGO                  YES, GOT IT
         L     R2,ASCBJBNS            GET # START/MOUNT/LOGON NAME
         CLC   Ø(8,R2),=C'PCAUTH  '   IS IT OUR ADDRESS SPACE ?
         BE    BINGO                  YES, GOT IT
RUNLOOP  DS    ØH
         LA    R11,4(,R11)            NEXT ASVT ENTRY
         BCT   R12,ASCBLOOP           CONTINUE TILL ASVTMAXU REACHED
         WTO   'PUXMSEØ3 PCAUTH not found.     ',ROUTCDE=11
         OI    #PGMFLAG,#PCANOTF      ADDRESS SPACE NOT FOUND FLAG
PRØ6ØØ   DS    ØH
         PR                           POP STACK AND RETURN TO CALLER
BINGO    DS    ØH                     It is our address space
         ST    R8,ASCB#               Save ASCB address for future use
         L     R9,ASCBASSB
         ST    R9,ASSB#               Save ASSB address for future use
         USING ASSB,R9
         L     R8,ASSBXMSE
         ST    R8,XMSE#               Save XMSE address for future use
         B     PRØ6ØØ
         DROP  R8
         DROP  R9
```

```
         EJECT
*********************************************************************
* This routine write the 1st title line on LISTXMSE.                *
*********************************************************************
WRITE_LINE1 DS ØH
         BAKR  R14,Ø           PUSH ENVIRONMENT INTO STACK
         MVC   HEX1,ASCB#      Convert to character
         BAS   R14,CONVERT_TO_CHAR
         MVC   £XMSPCAS,HEX2   Let's put it on output line
         MVC   HEX1,ASSB#      Convert to character
         BAS   R14,CONVERT_TO_CHAR
         MVC   £XMSPCSS,HEX2   Let's put it on output line
         MVC   HEX1,XMSE#      Convert to character
         BAS   R14,CONVERT_TO_CHAR
         MVC   £XMSPCXM,HEX2   Let's put it on output line
         MVC   £XMSLINE(XMSLIN1L),£XMSLIN1
         BAS   R14,WRITE_LISXMSE_LINE
         PR                    POP STACK AND RETURN TO CALLER
         EJECT
*********************************************************************
* This routine write the second title line on LISTXMSE.             *
*********************************************************************
WRITE_LINE2 DS ØH
         BAKR  R14,Ø           PUSH ENVIRONMENT INTO STACK
         MVC   £XMSLINE(XMSLIN2L),£XMSLIN2
         BAS   R14,WRITE_LISXMSE_LINE
         PR                    POP STACK AND RETURN TO CALLER
         EJECT
*********************************************************************
* This routine drives the logic for diving into PCAUTH's EPVT.      *
*********************************************************************
PROCESA_XMSE   DS ØH
         BAKR  R14,Ø             PUSH ENVIRONMENT INTO STACK
         USING ASCB,R11
         USING ASSB,R12
         L     R11,ASCB#         Address of target address space ASCB
         L     R12,ASCBASSB      Address of target address space ASSB
         L     R1Ø,ASSBXMSE      Address of 1st XMSE block
         MODESET KEY=ZERO,MODE=SUP
         BAS   R14,ALESERV_ADD   Get an ALET for the address space
         TM    #PGMFLAG,#ALETNOK ok ?
         BO    PR159357          No, just go out
         BAS   R14,CROSS_MEM     Let's do some cross-memory work
         BAS   R14,ALESERV_DEL   Delete access to other address space
PR159357 DS    ØH
         MODESET KEY=NZERO,MODE=PROB
         PR                      POP STACK AND RETURN TO CALLER
         EJECT
*********************************************************************
* This routine get an ALET for the target address space (PCAUTH in  *
```

```
* our case).                                                          *
*********************************************************************
ALESERV_ADD    DS ØH
         BAKR  R14,Ø                PUSH ENVIRONMENT INTO STACK
         ALESERV ADD,STOKEN=ASSBSTKN,ALET=MYALET,CHKEAX=NO
         LTR   R15,R15              Let's see rc
         BZ    PR147852             Ø, ok
         ST    R15,HEX1             Otherwise send a message
         BAS   R14,CONVERT_TO_CHAR
         MVC   WTO2+18(8),HEX2
         WTO   'PUXMSEØ4  unable to obtain ALET',ROUTCDE=11
WTO2     WTO   '          XXXXXXXX is the return code',ROUTCDE=11
         OI    #PGMFLAG,#ALETNOK    POSICIONA FLAG ANTES DE SALIR
PR147852 DS    ØH
         PR                         POP STACK AND RETURN TO CALLER
         EJECT
*********************************************************************
* This routine extracts the XMSE chain from the private part of      *
* PCAUTH address space. It executes in AR (access register) mode.    *
* R1Ø contains the 1st XMSE address.                                 *
*********************************************************************
CROSS_MEM      DS ØH
         BAKR  R14,Ø                PUSH ENVIRONMENT INTO STACK
         LAM   R1Ø,R1Ø,MYALET       Load the PCAUTH's ALET
         USING XMSE,R1Ø             Establish addressability to XMSE
         SAC   512                  Switch to AR mode
XMSELOOP DS    ØH
         CLC   XMSEACRO,=C'XMSE'    Good acronym ?
         BNE   BADACRO              No, problem.
         ST    R1Ø,HEX1             Convert the XMSE address
         BAS   R14,CONVERT_TO_CHAR  To character
         MVC   £XMSADDR,HEX2        Put on output line
         MVC   £XMSJBNA,XMSEJBNA    Job Name
         MVC   HEX1,XMSEASID        Convert the asid number
         BAS   R14,CONVERT_TO_CHAR  To character
         MVC   £XMSASID(4),HEX2     Put on output line
         MVC   HEX1,XMSEPREV        Convert address of previous XMSE
         BAS   R14,CONVERT_TO_CHAR  To character
         MVC   £XMSPREV,HEX2        Put on output line
         MVC   HEX1,XMSENEXT        Convert address of next XMSE
         BAS   R14,CONVERT_TO_CHAR  To character
         MVC   £XMSNEXT,HEX2        Put on output line
         LH    R8,XMSEASID          Take the ASID number and
*                                   go in the ASVT to see
         BAS   R14,CHECK_ASVT       if the slot is lost
         MVC   HEX1,XMSESETC        Convert the SETC address
         BAS   R14,CONVERT_TO_CHAR  To character
         MVC   £XMSSETC,HEX2        Put on the output line
         BAS   R14,CHECK_SETC       Let's see SETC detail
         SAC   Ø                    Go back into home mode
```

```
         BAS   R14,WRITE_LISXMSE_LINE
         SAC   512                 Switch again into AR mode
         L     R1Ø,XMSENEXT        Let's see next XMSE
         LTR   R1Ø,R1Ø             Valid ? (not x'ØØØØØØØØ') ?
         BNZ   XMSELOOP            Yes, loop
*                                  No, job done
         SAC   Ø                   Go back into home mode
         PR                        POP STACK AND RETURN TO CALLER
BADACRO  DS    ØH
         SAC   Ø                   Go back into home mode
         WTO   'PUXMSEØ5  Problem with XMSE chain into  ',ROUTCDE=11
         WTO   'PUXMSEØ5  the PCAUTH EPVT.              ',ROUTCDE=11
         OI    #PGMFLAG,#BADACRO   Flag on
         PR                        POP STACK AND RETURN TO CALLER
         EJECT
*********************************************************************
* This routine checks in the ASVT if the slot corresponding to the  *
* asid number (in register 8) is lost or not.                       *
*********************************************************************
CHECK_ASVT     DS ØH
         BAKR  R14,Ø                 PUSH ENVIRONMENT INTO STACK
         L     R9,16                 Get CVT address
         USING CVT,R9                Establish addressability
         L     R9,CVTASVT            Get ASVT address
         USING ASVT,R9               Establish addressability
         LA    R9,ASVTENTY           Get # of first entry
         BCTR  R8,Ø                  -1 on R8
         SLL   R8,2                  jump in the ASVT
*                                    (= (asid number -1) * 4)
         LA    R9,Ø(R8,R9)           We have the AZSVT slot
         MVC   HEX1,Ø(R9)
         BAS   R14,CONVERT_TO_CHAR
         MVC   £XMSASCB,HEX2
         CLC   Ø(4,R9),MASTASVT      Unavailable entry ?
         BNE   PR111111
         MVC   £XMSPERD,=C'** lost  **'
         L     R1Ø,LOST
         LA    R1Ø,1(,R1Ø)
         ST    R1Ø,LOST
PR111111 PR                      POP STACK AND RETURN TO CALLER
         EJECT
         DROP  R9
*********************************************************************
* This routine checks the SETC to determine which kind of cross-    *
* memory connection for this XMSE and how many connection do we have.*
* R1Ø contains the current XMSE address.                            *
* We are in AR mode into the PCAUTH private area.                   *
*********************************************************************
CHECK_SETC     DS ØH
         BAKR  R14,Ø                 PUSH ENVIRONMENT INTO STACK
```

```
         CPYA  R9,R1Ø                 Copy the ALET into R9
         L     R9,XMSESETC           Lets get SETC address
         USING SETC,R9               Establish addressability
         TM    SETCFLG1,SYSTEMLX     System LX ?
         BZ    NOSYSLX               No, something else
         MVC   £XMSSYLX,=C'SysLX'
NOSYSLX  DS    ØH
         MVC   HEX1,SETCTO           See num of to/from connections
         BAS   R14,CONVERT_TO_CHAR
         MVC   £XMSTOFR,HEX2
         PR                          POP STACK AND RETURN TO CALLER
         EJECT
         DROP  R9
*********************************************************************
* This routine frees the ALET.                                      *
*********************************************************************
ALESERV_DEL    DS ØH
         BAKR  R14,Ø             PUSH ENVIRONMENT INTO STACK
         ALESERV DELETE,ALET=MYALET,CHKEAX=NO
         PR                      POP STACK AND RETURN TO CALLER
         EJECT
*********************************************************************
* This routine writes the last lines of total on LISTXMSE.          *
*********************************************************************
WRITE_TOTAL DS ØH
         BAKR  R14,Ø               PUSH ENVIRONMENT INTO STACK
         MVC   HEXWORD,MAXU        Convert ASVTMAXU into decimal
         BAS   R14,CONVERT_TO_DEC
         MVC   £XMSMAXU,DECWORD    Put it into output line
         MVC   £XMSLINE(XMSLIN3L),£XMSLIN3
         BAS   R14,WRITE_LISXMSE_LINE
         MVC   HEXWORD,MAXI        Convert ASVTMAXI into decimal
         BAS   R14,CONVERT_TO_DEC
         MVC   £XMSMAXI,DECWORD    Put it into output line
         MVC   £XMSLINE(XMSLIN4L),£XMSLIN4
         BAS   R14,WRITE_LISXMSE_LINE
         MVC   HEXWORD,LOST        Convert num lost ent into decimal
         BAS   R14,CONVERT_TO_DEC
         MVC   £XMSLOST,DECWORD    Put it into output line
         MVC   £XMSLINE(XMSLIN5L),£XMSLIN5
         BAS   R14,WRITE_LISXMSE_LINE
         PR                        POP STACK AND RETURN TO CALLER
         EJECT
*********************************************************************
* This routine writes a line on LISTXMSE and reinits current line.  *
*********************************************************************
WRITE_LISXMSE_LINE DS ØH
         BAKR  R14,Ø           PUSH ENVIRONMENT INTO STACK
         PUT   LISTXMSE,£XMSLINE
         MVI   £XMSLINE,C' '
         MVC   £XMSLINE+1(L'£XMSLINE-1),£XMSLINE
```

```
         PR                    POP STACK AND RETURN TO CALLER
         EJECT
*********************************************************************
* This routine closes all DCBs.                                     *
*********************************************************************
CLOSDCBS DS    ØH
         BAKR  R14,Ø           PUSH ENVIRONMENT INTO STACK
         CLOSE (LISTXMSE)
         PR                    POP STACK AND RETURN TO CALLER
         EJECT
*********************************************************************
* This routine translates hexadecimal into printable format.
*********************************************************************
CONVERT_TO_CHAR DS ØH
         BAKR  R14,Ø                 PUSH ENVIRONMENT INTO STACK
         XR    R9,R9                 Clear R9
         IC    R9,HEX1               LOAD FIRST BYTE
         SRL   R9,4                  ELIMINATE 4 RIGHT MOST BITS
         STC   R9,HEX2               SAVE FIRST 4 BITS
         IC    R9,HEX1               LOAD FIRST BYTE
         SLL   R9,28                 ELIMINATE 4 LEFT MOST BITS
         SRL   R9,28
         STC   R9,HEX2+1             SAVE SECOND SET OF 4 BITS
         IC    R9,HEX1+1             LOAD SECOND BYTE
         SRL   R9,4                  ELIMINATE 4 RIGHT MOST BITS
         STC   R9,HEX2+2             SAVE THIRD SET OF 4 BITS
         IC    R9,HEX1+1             LOAD SECOND BYTE
         SLL   R9,28                 ELIMINATE 4 LEFT MOST BITS
         SRL   R9,28
         STC   R9,HEX2+3             SAVE FOURTH SET OF 4 BITS
         IC    R9,HEX1+2             LOAD THIRD  BYTE
         SRL   R9,4                  ELIMINATE 4 RIGHT MOST BITS
         STC   R9,HEX2+4             SAVE FIFTH  SET OF 4 BITS
         IC    R9,HEX1+2             LOAD THIRD  BYTE
         SLL   R9,28                 ELIMINATE 4 LEFT MOST BITS
         SRL   R9,28
         STC   R9,HEX2+5             SAVE SIXTH  SET OF 4 BITS
         IC    R9,HEX1+3             LOAD FOURTH BYTE
         SRL   R9,4                  ELIMINATE 4 RIGHT MOST BITS
         STC   R9,HEX2+6             SAVE SEVENTH SET OF 4 BITS
         IC    R9,HEX1+3             LOAD FOURTH BYTE
         SLL   R9,28                 ELIMINATE 4 LEFT MOST BITS
         SRL   R9,28
         STC   R9,HEX2+7             SAVE EIGHTH SET OF 4 BITS
         TR    HEX2(L'HEX2),TRTAB    TRANSLATE TO PRINTABLE CHAR
         XC    HEX1,HEX1             CLEAR FIELD
         PR                          POP STACK AND RETURN TO CALLER
         EJECT
*********************************************************************
* This routine converts a binary field into decimal printable charact*
* - Input  : field 'HEXWORD' (binary full word)                     *
```

```
* - Output : field 'DECWORD' (4 char)                                  *
*********************************************************************
CONVERT_TO_DEC DS ØH
         BAKR  R14,Ø           PUSH ENVIRONMENT INTO STACK
         L     RØ,HEXWORD      LOAD BINARY FIELD INTO REGISTER
         CVD   RØ,DWORD        CONVERT IT TO PACKED DECIMAL
         OI    DWORD+7,X'ØF'   CLEAR SIGN BIT
         UNPK  DECWORD,DWORD   UNPACK THIS STAFF INTO OUTPUT FIELD
         PR                    POP STACK AND RETURN TO CALLER
         EJECT
*********************************************************************
* This routine checks RC, restores registers and returns control.
*********************************************************************
RETURN   DS    ØH
         LA    R15,24              INIT R15
         TM    #PGMFLAG,#NOTAUTH   CHECK NOT AUTHORIZED FLAG
         BO    EXIT                IF SET, EXIT WITH RC=24
         LA    R15,2Ø              INIT R15
         TM    #PGMFLAG,#OPENERR   CHECK OPEN ERROR FLAG
         BO    EXIT                IF SET, EXIT WITH RC=2Ø
         LA    R15,16              INIT R15
         TM    #PGMFLAG,#PCANOTF   CHECK PCAUTH FLAG
         BO    EXIT                IF SET, EXIT WITH RC=16
         LA    R15,12              INIT R15
         TM    #PGMFLAG,#ALETNOK   CHECK ALET NOT OK FLAG
         BO    EXIT                IF SET, EXIT WITH RC=12
         LA    R15,8               INIT R15
         TM    #PGMFLAG,#BADACRO   CHECK BAD ACRONYM FLAG
         BO    EXIT                IF SET, EXIT WITH RC=8
         LA    R15,Ø               IF NOT, EXIT WITH RC=ØØ
EXIT     RCNTL RC=(15)
         EJECT
*
*
         TITLE 'PUXMSE literals.'
**
** Literals.
**
         LTORG
         EJECT ,
         TITLE 'PUXMSE Module Workarea'
*
TRTAB    DC   X'FØF1F2F3F4F5F6F7F8F9' CHARACTERS Ø123456789
         DC   X'C1C2C3C4C5C6'                   ABCDEF
ASCB#    DS    F
ASSB#    DS    F
XMSE#    DS    F
MYALET   DS    F
MASTASVT DS    F
HEX1     DS    F
HEX2     DS    D
```

```
MAXI     DS    F
MAXU     DS    F
LOST     DC    F'Ø'
HEXWORD  DS    F
DECWORD  DS    CL4
DWORD    DS    D
#PGMFLAG DC    B'ØØØØØØØØ'     Flag used for internal logic
#NOTAUTH EQU   B'1ØØØØØØØ'     Not authorized program
#OPENERR EQU   B'Ø1ØØØØØØ'     Error opening LISTXMSE
#PCANOTF EQU   B'ØØ1ØØØØØ'     We didn't find PCAUTH
#ALETNOK EQU   B'ØØØ1ØØØØ'     We didn't get the ALET (cross-mem)
#BADACRO EQU   B'ØØØØ1ØØØ'     Problem in scanning the XMSE chain
#HIGHON  EQU   B'1ØØØØØØØ'     HIGH BIT ON
*
*********************************************************************
* Print lines definitions.                                          *
*********************************************************************
£XMSLIN1 DS  ØH
         DC  C'1 PCAUTH ASCB# '
£XMSPCAS DS  CL8
         DC  C' ASSB# '
£XMSPCSS DS  CL8
         DC  C' XMSE# '
£XMSPCXM DS  CL8
XMSLIN1L EQU *-£XMSLIN1
£XMSLIN2 DS  ØH
         DC  C'ØXmseAddr Job Name Asid XmsePrev XmseNext AscbAddr     '
         DC  C'        XmseSetc       To  From'
XMSLIN2L EQU *-£XMSLIN2
£XMSLIN3 DS  ØH
         DC  C'1 Number of slots in the ASVT           '
£XMSMAXU DS  CL4
XMSLIN3L EQU *-£XMSLIN3
£XMSLIN4 DS  ØH
         DC  C'  Max users from Parmlib (IEASYSxx)     '
£XMSMAXI DS  CL4
XMSLIN4L EQU *-£XMSLIN4
£XMSLIN5 DS  ØH
         DC  C'  Number of lost entries in the ASVT    '
£XMSLOST DS  CL4
XMSLIN5L EQU *-£XMSLIN5
£XMSLINE DC  CL133' '
£XMSASA  EQU £XMSLINE,1
£XMSADDR EQU £XMSASA+1,8
£XMSSEP1 EQU £XMSADDR+8,1
£XMSJBNA EQU £XMSSEP1+1,8
£XMSSEP2 EQU £XMSJBNA+8,1
£XMSASID EQU £XMSSEP2+1,4
£XMSSEP3 EQU £XMSASID+4,1
£XMSPREV EQU £XMSSEP3+1,8
£XMSSEP4 EQU £XMSPREV+8,1
```

```
*********************************************************************
* This routine get an ALET for the target address space (PCAUTH in   *
* our case).                                                         *
*********************************************************************
ALESERV_ADD    DS ØH
         BAKR  R14,Ø               PUSH ENVIRONMENT INTO STACK
         ALESERV ADD,STOKEN=ASSBSTKN,ALET=MYALET,CHKEAX=NO
         LTR   R15,R15             Let's see rc
         BZ    PR147852            Ø, OK
         ST    R15,HEX1            Otherwise send a message
         BAS   R14,CONVERT_TO_CHAR
         MVC   WTO2+18(8),HEX2
         WTO   'PUXMSEØ4  unable to obtain ALET',ROUTCDE=11
WTO2     WTO   '          XXXXXXXX is the return code',ROUTCDE=11
         OI    #PGMFLAG,#ALETNOK   POSICIONA FLAG ANTES DE SALIR
PR147852 DS    ØH
         PR                        POP STACK AND RETURN TO CALLER
         EJECT
*********************************************************************
* This routine extracts the XMSE chain from the private part of      *
* PCAUTH address space. It executes in AR (access register) mode.    *
* R1Ø contains the first XMSE address.                               *
*********************************************************************
CROSS_MEM      DS ØH
         BAKR  R14,Ø               PUSH ENVIRONMENT INTO STACK
         LAM   R1Ø,R1Ø,MYALET      Load the PCAUTH's ALET
         USING XMSE,R1Ø            Establish addressability to XMSE
         SAC   512                 Switch to AR mode
XMSELOOP DS    ØH
         CLC   XMSEACRO,=C'XMSE'   Good acronym?
         BNE   BADACRO             No, problem.
         ST    R1Ø,HEX1            Convert the XMSE address
         BAS   R14,CONVERT_TO_CHAR To character
         MVC   £XMSADDR,HEX2       Put on output line
         MVC   £XMSJBNA,XMSEJBNA   Job Name
         MVC   HEX1,XMSEASID       Convert the asid number
         BAS   R14,CONVERT_TO_CHAR To character
         MVC   £XMSASID(4),HEX2    Put on output line
         MVC   HEX1,XMSEPREV       Convert address of previous XMSE
         BAS   R14,CONVERT_TO_CHAR To character
         MVC   £XMSPREV,HEX2       Put on output line
         MVC   HEX1,XMSENEXT       Convert address of next XMSE
         BAS   R14,CONVERT_TO_CHAR To character
         MVC   £XMSNEXT,HEX2       Put on output line
         LH    R8,XMSEASID         Take the ASID number and
*                                  go in the ASVT to see
         BAS   R14,CHECK_ASVT      if the slot is lost
         MVC   HEX1,XMSESETC       Convert the SETC address
         BAS   R14,CONVERT_TO_CHAR To character
         MVC   £XMSSETC,HEX2       Put on the output line
         BAS   R14,CHECK_SETC      Let's see SETC detail
```

```
         SAC   Ø                    Go back into home mode
         BAS   R14,WRITE_LISXMSE_LINE
         SAC   512                  Switch again into AR mode
         L     R1Ø,XMSENEXT         Let's see next XMSE
         LTR   R1Ø,R1Ø              Valid ? (not x'ØØØØØØØØ') ?
         BNZ   XMSELOOP             Yes, loop
*                                   No, job done
         SAC   Ø                    Go back into home mode
         PR                         POP STACK AND RETURN TO CALLER
BADACRO  DS    ØH
         SAC   Ø                    Go back into home mode
         WTO   'PUXMSEØ5  Problem with XMSE chain into  ',ROUTCDE=11
         WTO   'PUXMSE Ø5  the PCAUTH EPVT.             ',ROUTCDE=11
         OI    #PGMFLAG,#BADACRO    Flag on
         PR                         POP STACK AND RETURN TO CALLER
         EJECT
*********************************************************************
* This routine checks in the ASVT if the slot corresponding to the  *
* asid number (in register 8) is lost or not.                       *
*********************************************************************
CHECK_ASVT     DS ØH
         BAKR  R14,Ø                  PUSH ENVIRONMENT INTO STACK
         L     R9,16                  Get CVT address
         USING CVT,R9                 Establish addressability
         L     R9,CVTASVT             Get ASVT address
         USING ASVT,R9                Establish addressability
         LA    R9,ASVTENTY            Get # of first entry
         BCTR  R8,Ø                   -1 on R8
         SLL   R8,2                   jump in the ASVT
*                                     (= (asid number -1) * 4)
         LA    R9,Ø(R8,R9)            We have the AZSVT slot
         MVC   HEX1,Ø(R9)
         BAS   R14,CONVERT_TO_CHAR
         MVC   £XMSASCB,HEX2
         CLC   Ø(4,R9),MASTASVT       Unavailable entry ?
         BNE   PR111111
         MVC   £XMSPERD,=C'** lost  **'
         L     R1Ø,LOST
         LA    R1Ø,1(,R1Ø)
         ST    R1Ø,LOST
PR111111 PR                      POP STACK AND RETURN TO CALLER
         EJECT
         DROP  R9
*********************************************************************
* This routine checks the SETC to determine which kind of cross-    *
* memory connection for this XMSE and how many connection do we have.*
* R1Ø contains the current XMSE address.                            *
* We are in AR mode into the PCAUTH private area.                   *
*********************************************************************
CHECK_SETC     DS ØH
         BAKR  R14,Ø                  PUSH ENVIRONMENT INTO STACK
```

```
         CPYA  R9,R1Ø                Copy the ALET into R9
         L     R9,XMSESETC           Lets get SETC address
         USING SETC,R9               Establish addressability
         TM    SETCFLG1,SYSTEMLX     System LX ?
         BZ    NOSYSLX               No, something else
         MVC   £XMSSYLX,=C'SysLX'
NOSYSLX  DS    ØH
         MVC   HEX1,SETCTO           See num of to/from connections
         BAS   R14,CONVERT_TO_CHAR

         MVC   £XMSTOFR,HEX2
         PR                          POP STACK AND RETURN TO CALLER
         EJECT
         DROP  R9
*********************************************************************
* This routine frees the ALET.                                      *
*********************************************************************
ALESERV_DEL    DS ØH
         BAKR  R14,Ø             PUSH ENVIRONMENT INTO STACK
         ALESERV DELETE,ALET=MYALET,CHKEAX=NO
         PR                      POP STACK AND RETURN TO CALLER
         EJECT
*********************************************************************
* This routine writes the last lines of total on LISTXMSE.          *
*********************************************************************
WRITE_TOTAL DS ØH
         BAKR  R14,Ø               PUSH ENVIRONMENT INTO STACK
         MVC   HEXWORD,MAXU        Convert ASVTMAXU into decimal
         BAS   R14,CONVERT_TO_DEC

         MVC   £XMSMAXU,DECWORD    Put it into output line
         MVC   £XMSLINE(XMSLIN3L),£XMSLIN3
         BAS   R14,WRITE_LISXMSE_LINE
         MVC   HEXWORD,MAXI        Convert ASVTMAXI into decimal
         BAS   R14,CONVERT_TO_DEC
         MVC   £XMSMAXI,DECWORD    Put it into output line
         MVC   £XMSLINE(XMSLIN4L),£XMSLIN4
         BAS   R14,WRITE_LISXMSE_LINE
         MVC   HEXWORD,LOST        Convert num lost ent into decimal
         BAS   R14,CONVERT_TO_DEC
         MVC   £XMSLOST,DECWORD    Put it into output line
         MVC   £XMSLINE(XMSLIN5L),£XMSLIN5
         BAS   R14,WRITE_LISXMSE_LINE
         PR                        POP STACK AND RETURN TO CALLER
         EJECT
*********************************************************************
* This routine writes a line on LISTXMSE and reinits current line.  *
*********************************************************************
WRITE_LISXMSE_LINE DS ØH
         BAKR  R14,Ø           PUSH ENVIRONMENT INTO STACK
         PUT   LISTXMSE,£XMSLINE
         MVI   £XMSLINE,C' '
```

```
         MVC   £XMSLINE+1(L'£XMSLINE-1),£XMSLINE
         PR                    POP STACK AND RETURN TO CALLER
         EJECT
*************************************************************************
* This routine closes all DCBs.                                         *
*************************************************************************
CLOSDCBS DS    ØH
         BAKR  R14,Ø           PUSH ENVIRONMENT INTO STACK
         CLOSE (LISTXMSE)
         PR                    POP STACK AND RETURN TO CALLER
EJECT
*************************************************************************
* This routine translates hexadecimal into printable format.
*************************************************************************
CONVERT_TO_CHAR DS ØH
         BAKR  R14,Ø                   PUSH ENVIRONMENT INTO STACK
         XR    R9,R9                   Clear R9
         IC    R9,HEX1                 LOAD FIRST BYTE
         SRL   R9,4                    ELIMINATE 4 RIGHT MOST BITS
         STC   R9,HEX2                 SAVE FIRST 4 BITS
         IC    R9,HEX1                 LOAD FIRST BYTE
         SLL   R9,28                   ELIMINATE 4 LEFT MOST BITS
         SRL   R9,28
         STC   R9,HEX2+1               SAVE SECOND SET OF 4 BITS
         IC    R9,HEX1+1               LOAD SECOND BYTE
         SRL   R9,4                    ELIMINATE 4 RIGHT MOST BITS
         STC   R9,HEX2+2               SAVE THIRD SET OF 4 BITS
         IC    R9,HEX1+1               LOAD SECOND BYTE
         SLL   R9,28                   ELIMINATE 4 LEFT MOST BITS
         SRL   R9,28
         STC   R9,HEX2+3               SAVE FOURTH SET OF 4 BITS
         IC    R9,HEX1+2               LOAD THIRD  BYTE
         SRL   R9,4                    ELIMINATE 4 RIGHT MOST BITS
         STC   R9,HEX2+4               SAVE FIFTH  SET OF 4 BITS
         IC    R9,HEX1+2               LOAD THIRD  BYTE
         SLL   R9,28                   ELIMINATE 4 LEFT MOST BITS
         SRL   R9,28
         STC   R9,HEX2+5               SAVE SIXTH  SET OF 4 BITS
         IC    R9,HEX1+3               LOAD FOURTH BYTE
         SRL   R9,4                    ELIMINATE 4 RIGHT MOST BITS
         STC   R9,HEX2+6               SAVE SEVENTH SET OF 4 BITS
         IC    R9,HEX1+3               LOAD FOURTH BYTE
         SLL   R9,28                   ELIMINATE 4 LEFT MOST BITS
         SRL   R9,28
         STC   R9,HEX2+7               SAVE EIGHTH SET OF 4 BITS
         TR    HEX2(L'HEX2),TRTAB      TRANSLATE TO PRINTABLE CHAR
         XC    HEX1,HEX1               CLEAR FIELD
         PR                            POP STACK AND RETURN TO CALLER
         EJECT
*************************************************************************
* This routine converts a binary field into decimal printable charact*
```

```
* - Input  : field 'HEXWORD' (binary full word)                      *
* - Output : field 'DECWORD' (4 char)                                *
*********************************************************************
CONVERT_TO_DEC DS ØH
         BAKR  R14,Ø           PUSH ENVIRONMENT INTO STACK
         L     RØ,HEXWORD      LOAD BINARY FIELD INTO REGISTER
         CVD   RØ,DWORD        CONVERT IT TO PACKED DECIMAL
         OI    DWORD+7,X'ØF'   CLEAR SIGN BIT
         UNPK  DECWORD,DWORD   UNPACK THIS STAFF INTO OUTPUT FIELD
         PR                    POP STACK AND RETURN TO CALLER
         EJECT
*********************************************************************
* This routine checks RC, restores registers and returns control.
*********************************************************************
RETURN   DS    ØH
         LA    R15,24               INIT R15
         TM    #PGMFLAG,#NOTAUTH    CHECK NOT AUTHORIZED FLAG
         BO    EXIT                 IF SET, EXIT WITH RC=24
         LA    R15,2Ø               INIT R15
         TM    #PGMFLAG,#OPENERR    CHECK OPEN ERROR FLAG
         BO    EXIT                 IF SET, EXIT WITH RC=2Ø
         LA    R15,16               INIT R15
         TM    #PGMFLAG,#PCANOTF    CHECK PCAUTH FLAG
         BO    EXIT                 IF SET, EXIT WITH RC=16
         LA    R15,12               INIT R15
         TM    #PGMFLAG,#ALETNOK    CHECK ALET NOT OK FLAG
         BO    EXIT                 IF SET, EXIT WITH RC=12
         LA    R15,8                INIT R15
         TM    #PGMFLAG,#BADACRO    CHECK BAD ACRONYM FLAG
         BO    EXIT                 IF SET, EXIT WITH RC=8
         LA    R15,Ø                IF NOT, EXIT WITH RC=ØØ
EXIT     RCNTL RC=(15)
         EJECT
*
*
         TITLE 'PUXMSE literals.'
**
** Literals.
**
         LTORG
         EJECT ,
         TITLE 'PUXMSE Module Workarea'
*
TRTAB    DC   X'FØF1F2F3F4F5F6F7F8F9' CHARACTERS Ø123456789
         DC   X'C1C2C3C4C5C6'                   ABCDEF
ASCB#    DS    F
ASSB#    DS    F
XMSE#    DS    F
MYALET   DS    F
MASTASVT DS    F
HEX1     DS    F
```

```
HEX2     DS    D
MAXI     DS    F
MAXU     DS    F
LOST     DC    F'Ø'
HEXWORD  DS    F
DECWORD  DS    CL4
DWORD    DS    D
#PGMFLAG DC    B'ØØØØØØØØ'     Flag used for internal logic
#NOTAUTH EQU   B'1ØØØØØØØ'     Not authorized program
#OPENERR EQU   B'Ø1ØØØØØØ'     Error opening LISTXMSE
#PCANOTF EQU   B'ØØ1ØØØØØ'     We didn't find PCAUTH
#ALETNOK EQU   B'ØØØ1ØØØØ'     We didn't get the ALET (cross-mem)
#BADACRO EQU   B'ØØØØ1ØØØ'     Problem in scanning the XMSE chain
#HIGHON  EQU   B'1ØØØØØØØ'     HIGH BIT ON
*********************************************************************
* Print lines definitions.                                          *
*********************************************************************
£XMSLIN1 DS  ØH
         DC  C'1 PCAUTH ASCB# '
£XMSPCAS DS  CL8
         DC  C' ASSB# '
£XMSPCSS DS  CL8
         DC  C' XMSE# '
£XMSPCXM DS  CL8
XMSLIN1L EQU *-£XMSLIN1
£XMSLIN2 DS  ØH
         DC  C'ØXmseAddr Job Name Asid XmsePrev XmseNext AscbAddr     '
         DC  C'        XmseSetc       To  From'
XMSLIN2L EQU *-£XMSLIN2
£XMSLIN3 DS  ØH
         DC  C'1 Number of slots in the ASVT           '
£XMSMAXU DS  CL4
XMSLIN3L EQU *-£XMSLIN3
£XMSLIN4 DS  ØH
         DC  C'  Max users from Parmlib (IEASYSxx)     '
£XMSMAXI DS  CL4
XMSLIN4L EQU *-£XMSLIN4
£XMSLIN5 DS  ØH
         DC  C'  Number of lost entries in the ASVT    '
£XMSLOST DS  CL4
XMSLIN5L EQU *-£XMSLIN5
£XMSLINE DC  CL133' '
£XMSASA  EQU £XMSLINE,1
£XMSADDR EQU £XMSASA+1,8
£XMSSEP1 EQU £XMSADDR+8,1
£XMSJBNA EQU £XMSSEP1+1,8
£XMSSEP2 EQU £XMSJBNA+8,1
£XMSASID EQU £XMSSEP2+1,4
£XMSSEP3 EQU £XMSASID+4,1
£XMSPREV EQU £XMSSEP3+1,8
£XMSSEP4 EQU £XMSPREV+8,1
```

```
£XMSNEXT EQU £XMSSEP4+1,8
£XMSSEP5 EQU £XMSNEXT+8,1
£XMSASCB EQU £XMSSEP5+1,8
£XMSSEP6 EQU £XMSASCB+8,1
£XMSPERD EQU £XMSSEP6+1,11
£XMSSEP7 EQU £XMSPERD+11,1
£XMSSETC EQU £XMSSEP7+1,8
£XMSSEP8 EQU £XMSSETC+8,1
£XMSSYLX EQU £XMSSEP8+1,5
£XMSSEP9 EQU £XMSSYLX+5,1
£XMSTOFR EQU £XMSSEP9+1,8
*********************************************************************
* THE DCBS                                                          *
*********************************************************************
         DS    ØH
LISTXMSE DCB   DDNAME=LISTXMSE,MACRF=(PM),RECFM=FBA,LRECL=133,
               DSORG=PS
         EJECT
*********************************************************************
* Dsects                                                            *
*********************************************************************
XMSE     DSECT
XMSEACRO DS    CL4               Acronym 'XMSE'
XMSESETC DS    F                 SETC address
         DS    CL8
XMSEPREV DS    F                 Previous block
XMSENEXT DS    F                 Next block
         DS    CL4
XMSEJBNA DS    CL8               Job Name
XMSEASID DS    CL2               Asid hexa
SETC     DSECT
SETCACRO DS    CL4               Acronym 'SETC'
         DS    CL2
SETCFLG1 DS    X                 Flogs
SYSTEMLX EQU   B'10000000'       Flag for System LX
         DS    X
         DS    CL12
SETCTO   DS    H                 Number of 'TO'   connections
SETCFROM DS    H                 Number of 'FROM' connections
         CVT      DSECT=YES,LIST=YES
         EJECT
         IHAASVT  LIST=YES
         EJECT
         IHAASCB  LIST=YES
         EJECT
         IHAASSB LIST=YES
         DCBD     DEVD=(DA,TA),DSORG=(QS,BS)
         END
```

Michel Joly
Systems Programmer (France)

z/OS managed system infrastructure for setup

z/OS managed system infrastructure (msys) for setup is delivered as part of z/OS Release 1, and further support is provided in z/OS Release 2. msys is essentially a system wizard for deploying z/OS and its associated functions. It uses 'smart configuration dialogs' to help users through the setup process. The term 'wizard' evokes a great deal of stigma in the large systems world, because it is essentially a technology that has become synonymous with PCs (and their animated paperclips) and there is an underlying feeling that using system wizards is an indication of poorly trained sysprogs. However, we found that using msys for setup during the installation of z/OS and some of its component resulted in massive time savings.

There are currently two types ofwizards available for z/OS – *planning* wizards and *customization* wizards. Planning wizards produce a customized view of documentation based on the answers that users provide for on-screen questions. Planning wizards provide easier access to the information that users need most whilst hiding that which is not required. However, unlike the wizards found on PCs, z/OS wizards do not directly change anything on the z/OS system because they are not directly connected to it.

Customization wizards provide tailored jobs, commands and information to help users complete a customization task. The most important customization wizard is msys for setup. It uses workstation-based dialogs for specifying a system configuration. The dialogs understand the current system configuration and take it into account. An automated process then updates the system with the defined configuration. This is centred around a z/OS management directory, which becomes the central location for all configuration values. Unlike the Web-based wizards msys for setup is able to execute updates and changes because it is an integral component of z/OS.

To quote IBM, "many customers are concerned about the lack of system programmer skill in the next few years, because many S/390 system programmers will retire in the not too distant future. It is quite difficult (and expensive) to fill the slots appropriately. msys for Setup addresses these issue by reducing the skill requirements needed for configuring z/OS and z/OS products." We found that the use of msys

made our experienced system programmers more productive. We found that configuration activities that typically take days when done manually can now be done within a day using msys for setup. With z/OS Release 2, additional z/OS components will use msys for setup for their configuration. And there are further benefits.

As a mainframer I shudder at the thought of dealing with TCP/IP and other networking integration issues. Unfortunately our networking staff do not all have the necessary mainframe experience required to help. Therefore, mainframe networking issues can take considerable engineer time to sort out. This is one area where we found msys invaluable. TCP/IP services can exploit msys for setup to define basic TCP/IP settings, in addition to rapid definition of network devices and links, and for the setup of FTP or Telnet 3270 servers. The corresponding TCP/IP datasets (such as tcp.data, and profile.tcp) are created automatically.

In z/OS Version 1 Release 1, the msys for setup framework is used by Parallel Sysplex functions to set up a Parallel Sysplex resource sharing environment. msys for setup can reduce tasks such as the setup of the system logger, which includes the logger requirements for IBM License Manager, for OPERLOG, and for LOGREC, as well as defining the options for enhanced catalog sharing. msys for setup will understand any existing definitions, so that they can be easily managed using the msys for setup dialogs.

With z/OS Release 2 this support is extended so that users will also be able to set up a Base Sysplex environment using msys for setup. It will have all functions of the Parallel Sysplex support relevant for Base Sysplex, including the support for migrating from a Base Sysplex environment to a Parallel Sysplex. msys for setup can also create the ISPF configuration table keyword file and load modules.

In summary, since the deployment of z/OS Version 1 Release 1 IBM has increased the use of wizards to ease system maintenance. System wizards can be either planning wizards or customization wizards. Currently IBM has planning wizards for installation planning, DFSMS migration, and planning for e-business. The configuration wizards include SDSF Parallel Sysplex, Base Sysplex, IP, and Unix System Services. In the future most customization wizards will become components of msys. However, in the short term, IBM is likely to continue producing Web-based wizards because these are not tied to

the bi-annual z/OS release cycle and so can be deployed as soon as they are produced. Users can look forward to a wide range of wizards in the near future.

The msys for setup framework comes as an integral part of z/OS, and the support mentioned here is shipped as part of the corresponding z/OS components. The time savings alone will justify the use of msys and other wizards.

z/VM Version 4 Release 2

IBM has released Version 4 Release 2 of its z/VM operating system, which includes a number of zSeries software exploitations, connectivity enhancements, and systems management improvements. Using virtualization software as a foundation, it has new functions and tools that exploit VM capabilities on the mainframe, allowing sites to virtualize processor, communications, storage, and I/O resources to help reduce the overhead of planning, purchasing, and installing new hardware to support new workloads.

Technology exploitation issues include Clear-key RSA support of the IBM PCI Cryptographic Accelerator when corresponding function is available from Linux on zSeries, improved I/O performance for guest DASD channel programs when data resides above 2GB, HiperSockets, a high-speed internal TCP/IP network, and OSA-Express Token Ring, guest coupling duplex support, and guest support for FICON CTCA. Connectivity enhancements include guest LAN support, a new TCP/IP server for mail accessibility using IMAP, and TCP/IP stack security.

Systems management improvements cover ease-of-use functions for managing Linux images and the ability to move configurations and data from Virtual Image Facility (VIF). It also enables a large number of Linux server images on a single zSeries 900 or S/390, which can be deployed on standard processor engines or IFL processor features.Further information can be found at the following URL: http://www.ibm.com/eserver/zseries/zvm

IBM has announced a new integrated security feature of the z/Architecture with a PCI Cryptographic Accelerator that provides the required performance of the complex RSA crypto operations used in the SSL protocol.

Meanwhile, with z/OS support available by the end of 2001, System-Managed Coupling Facility (CF) Structure Duplexing adds improved failure recovery capabilities, while helping to reduce the complexity of CF structure recovery.

The newly-designed FICON Express adapter with its faster internal bus structure and new connectors is the z900's latest implementation of the Fibre Channel architecture. FICON Express adds to the basic functions with increased bandwidth to further consolidate and simplify configurations.

FICON Channel-to-Channel (CTC) connectivity increases bandwidth between systems and can potentially allow consolidation of several channels.

High-speed interconnects for TCP/IP communication, HiperSockets, let TCP/IP traffic travel between partitions at memory speed rather than network speed. Also new is OSA-Express Token Ring, along with the OSA-Express Gigabit Ethernet, supporting communications within the server, between servers, and out to users.

Contact your local IBM representative for further information.

* * *

Xephon will be holding its annual *Mainframe Futures* conference at the Radisson SAS Portman Hotel in London, on 22-23 November 2001. *Mainframe Futures* is designed specifically for technical managers, systems programmers, strategic planners, and other system specialists at MVS/ESA, OS/390, and z/OS installations. Topics include z/OS functionality, zSeries directions, performance managemen for the zServert, mainframe software pricing, the Workload Licence charge, and many more.

The attendance fee for the conference is £930 plus £108.50 VAT. For further information, please telephone the registrar, Toni Brown, on (01635) 33823.

http://www.xephon.com/events

* * *

TIBCO Software has announced plans to support Linux on IBM's z900 and S/390 servers with its integration software, allowing it to enable its customers to cross-access existing applications running on both mainframe and Unix environments in real-time.

For further information contact:
TIBCO Software, 3165 Porter Drive, Palo Alto, CA 94304 USA.
Tel: 650 846 1000
Fax: 650 846 1005

TIBCO Software, 35 New Bridge Street, London, EC4V 6BW, UK
Tel: 020 7964 3700
Fax: 020 7964 3737

http://www.tibco.com

* * *