189

MVS update

June 2002

In this issue

3 JES2 z/OS 1.2 dynamic PROCLIB

5 Java demo files delivered with Java 2

8 A VSAM browse routine – part 2

22 Displaying multi-volume dataset details

37 Maintaining a DASD configuration – revisited

41 Overview of ISPF panel processing commands

57 The fastest way to get SMS DASD space information

72 MVS news

MVS Update

Published by

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

North American office

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

Subscriptions and back-issues

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

MVS Update on-line

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

Editor

Trevor Eddolls

Disclaimer

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

Contributions

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

Printed in England.

JES2 z/OS 1.2 dynamic PROCLIB

Who has ever been obliged to shut down JES2 in order to add a new PROCLIB dataset?

Starting with z/OS 1.2, you can now dynamically add, remove, or modify PROCLIBs datasets, making this operation non-disruptive.

Dynamic PROCLIB can override existing PROC*xx* DDs in the JES2 start PROC.

There are two ways to use dynamic PROCLIBs:

1 Using JES2 commands $ADD, $DEL, and $T.

2 Using the PROCLIB init statement in JES2PARM, thus completely eliminating the static PROCLIBs in JES2 PROC.

SAMPLE JES2 START-UP PROC

```
//JES2    PROC VERSION=2Ø,MEMBER=JES2PARM
//IEFPROC EXEC PGM=HASJES&VERSION,DPRTY=(15,15),TIME=144Ø
//PROCØØ   DD  DSN=SYS2.PROCLIB,DISP=SHR
//         DD  DSN=SYS2.PROIBM.PROCLIB,DISP=SHR
//         DD  DSN=SYS2.NONIBM.PROCLIB,DISP=SHR
//         DD  DSN=SYS1.PROCLIB,DISP=SHR
//PROCØ1   DD  DSN=SYS2.TSO.PROCLIB,DISP=SHR
//HASPPARM DD  DSN=SYS1.PARMLIB(&MEMBER),DISP=SHR
//HASPLIST DD  DDNAME=IEFRDER
```

PROC00 concatenation is for START and EXEC PROCs.

The PROC01 is dedicated to TSO logon PROCs as defined in JES2PARM:

```
JOBCLASS(TSU) PROCLIB=Ø1
```

EXAMPLE 1: ADDING A PROC

```
$ADD PROCLIB(PROCØØ),DD1=DSN=SYS2.FLM.PROCLIB
$HASP319 PROCLIB(PROCØØ) DD(1)=(DSNAME=SYS2.FLM.PROCLIB)
```

If the name in PROCLIB() matches a DDname in the JES2 start PROC,

this PROCLIB concatenation will be used instead of the one in the JES2 start PROC.

Sysplex note: you need to execute this command on all MAS members.

EXAMPLE 2: DISPLAYING DYNAMIC PROCLIBS

You cannot view PROCLIBs defined in the JES2 start PROC, only PROCLIB $ADDed are displayed.

```
$D PROCLIB
$HASP319 PROCLIB(PROC00)
$HASP319 PROCLIB(PROC00) DD(1)=(DSNAME=SYS2.FLM.PROCLIB)
```

EXAMPLE 3: DISPLAY WITH DEBUG KEYWORD

```
$D PROCLIB,DEBUG
$HASP319 PROCLIB(PROC00)
$HASP319 PROCLIB(PROC00) USECOUNT=1,DDNAME=SYS00002,
$HASP319 CREATED=2002.032,12:52:03.76,
$HASP319 DD(1)=(DSNAME=SYS2.FLM.PROCLIB)
```

EXAMPLE 4: MODIFYING AN EXISTING PROCLIB CONCATENATION

This adds a second PROCLIB to an existing concatenation:

```
$ADD PROCLIB(PROC02),DD1=DSN=SYS1.PROCLIB
$HASP319 PROCLIB(PROC02)    DD(1)=(DSNAME=SYS1.PROCLIB)

$T PROCLIB(PROC02),DD2=DSN=SYS2.PROCLIB
$HASP319 PROCLIB(PROC02)
$HASP319 PROCLIB(PROC02)    DD(1)=(DSNAME=SYS1.PROCLIB),
$HASP319                    DD(2)=(DSNAME=SYS2.PROCLIB)
```

EXAMPLE 5: DELETING A DYNAMICALLY ADDED PROCLIB

You cannot delete PROCLIBs defined in the JES2 start PROC:

```
$DEL PROCLIB(PROC00)
$HASP319 PROCLIB(PROC00)
$HASP319 PROCLIB(PROC00) DD(1)=(DSNAME=SYS2.FLM.PROCLIB) -
$HASP319 ELEMENT DELETED
```

$D PROCLIB,DEBUG can still display some information after the DELETE:

```
$HASP319 PROCLIB(PROC00) DELETED PROCLIB,USECOUNT=1,
$HASP319 DDNAME=SYS00004,
$HASP319 CREATED=2002.032,12:52:03.76,
$HASP319 DD(1)=(DSNAME=SYS2.FLM.PROCLIB)
```

Francois Le Maner
System Engineer (France)

Java demo files delivered with Java 2

We have just installed Java 2 Technology Edition on our OS/390 machine. We have completed the 'Apply' stage for SMP/E and now we want to check that everything works as expected before running the 'Accept'. The installation guide points us in the direction of IVPs (Installation Verification Programs).

Installation Verification Programs is a super title that, for example with the CICS installation, promises us some sort of batch or interactive program. There is, however, only one section, 'Activating IBM Developer Kit for OS/390, Java 2 Technology Edition', which contains no pretty screens, no moving parts, and no dialog. We followed the instructions:

- Use the following shell command:

  ```
  export PATH=/usr/lpp/java/IBM/J1.3/bin:$PATH
  ```

  which adds the java bin to the PATH description, and for future use we changed the system path variable as appropriate in */etc/profile*.

- After setting the path, enter the following shell command:

  ```
  java -version
  ```

  IBM Developer Kit for OS/390, Java 2 Technology Edition is successfully installed if the 'java' command returns a summary of available options and the version option returns the current version of the IBM Developer Kit for OS/390, Java 2 Technology Edition code.

OK it works!

Brilliant, I have had some boring tasks in the past, but this tops them all. By complete coincidence I saw in one of the installation job listings (OAJVI2) the list of expanded (PAX) directories and files, and amongst the directories was the following:

```
J1.3/demo
```

which on our site is */usr/lpp/java/IBM/J1.3/demo/* and has a whole load of directories and files packed underneath it – one of which is applets.

Great! Now we can really try the Java. All we need to do is find the appropriate documentation to run these demos.

Why do the folk at IBM make life so difficult sometimes?

Here's what we did to get the demos up and running.

First, because we naturally expected to access Java via a browser we altered the PATH, LIBPATH, JAVA_HOME, and CLASSPATH in the httpd.envars file for our http server to reflect the new address of Java.

Secondly, as we found out on our first attempt to get the demos to run, we needed to have a 'pass' rule in the httpd.conf file. Ours was then simply:

```
Pass   /usr/lpp/java/IBM/J1.3/demo/*    /usr/lpp/java/IBM/J1.3/demo/*
```

And, thirdly, we needed to restart the server.

With our second attempt, and using the URL http://os390r10/usr/lpp/java/IBM/J1.3/demo/applets/Clock/example1 (where OS/390 Release 10 is the TCP/IP name of our test OS/390 Version 2 Release 10 machine), we were up and running with a Java demo clock. Contained in the directory */usr/lpp/java/IBM/J1.3/demo/applets/* are 21 subdirectories of demos with one or more examples (accessed as example1, example2, etc) per demo directory.

A few words about each.

- Animator – four examples, none of which started under Netscape or Internet Explorer (ERROR).

- ArcTest – one example. You enter the start and end position in degrees and an arc is drawn and filled in if the fill option is used.
- BarChart – simple colourful example of a bar chart. Sadly no moving parts.
- Blink – this applet had problems starting with java.lang.NoClassDefFoundError
- CardTest – super demo of various forms (layouts) of click buttons.
- Clock – a simple digital/analog clock.
- DitherTest – I don't really know what it is, but it enables a superb mixing of the three base colours of blue, red, and green.
- DrawTest – something for the kids amongst us. Six different colours and the options of either lines or freehand to play with.
- Fractal – I'm sure it's a good demo for programming; however, as fractals go it's somewhat primitive.
- GraphicsTest – demo of five different shapes.
- GraphLayout – four examples of moving graphs. I'm sure the code for this is pretty interesting and can be used for good and simple effects.
- ImageMap – super example of a mapped image. The image contains seven different active areas, which trigger nine different actions, including sound.
- JumpingBox – a simple but effective game. The box moves randomly in response to the cursor and you have to try to click when the cursor is over the box. Also with sound.
- MoleculeViewer – three examples, none of which worked on my browsers. Internet Explorer had nothing to say and Netscape mentioned something about transferring data from the host, but after a couple of minutes waiting it still didn't do anything to impress the viewer.
- NervousText – a simple moving (text) banner.

- SimpleGraph – very simple indeed.
- SortDemo – super graphical demonstration of the sort methods: bubble sort, bi-directional bubble sort, and quicksort. The sort of thing one would have liked when studying Computing/IT.
- SpreadSheet – a simple spreadsheet.
- SymbolTest – displays unicode character ranges.
- TicTacToe – the simple but ever popular TicTacToe, but with sound.
- WireFrame – four examples, three of which brought 'Error in model: FileFormatException: Token[,{,], line 3' (line 1 under Netscape). The first example brought nothing.

Altogether a good collection of fairly simple examples, which, used as IVPs, make life a little less boring for the installation team. The collection also gives the newcomer a good chance to investigate simple code (contained in .java files), to demonstrate techniques, and to avoid confusion by keeping things simple. It is just a pity that the chaps at IBM did not document it that well.

Rolf Parker
Systems Programmer (UK)

A VSAM browse routine – part 2

This month we conclude the code for a VSAM browse routine that can be used in place of the standard ISRBRO.

```
         MACRO
         POPNEST &P1
         COPY  PPFGBLCØ
         LCLC  &SUFFIX
&SUFFIX  SETC  '&PF_NEST(&PF_NI)'(5,4)
         AIF   ('&PF_NEST(&PF_NI)'(5,4) EQ '&P1').GOOD
         MNOTE 8,'&SUFFIX MACRO AT SAME LEVEL AS &P1 TERMINATOR.'
.GOOD    ANOP
&PF_NI   SETA  &PF_NI-1
```

```
         AIF   (&PF_NI GE Ø).OK
         MNOTE 8,'NEGATIVE NEST STACK POINTER. CHECK NUMBER OF ENDS.'
.OK      MEND
*********************************************************************
         MACRO
         STKINS &P1,&P2,&P3,&P4,&P5,&P6
         COPY  PPFGBLCØ
         AIF    ('&P1(2)' EQ '').NOTSUBL
         AIF    ('&P1(6)' EQ '' OR '&P1(6)' EQ '&PF_LIND(&PF_LI)').OKSU+
               BL
         MNOTE 12,'TOO MANY OPERANDS INSIDE PARENTHESES'
         MEXIT
.OKSUBL  PUSHINS     (&P1(1),&P1(2),&P1(3),&P1(4),&P1(5),&P1(6))
         MEXIT
.NOTSUBL AIF   ('&P2' EQ '' OR '&P2' EQ 'OR' OR '&P2' EQ 'AND' OR '&P2'+
               EQ 'ORIF' OR '&P2' EQ 'ANDIF').SGLOPR
         AIF   ('&P5' EQ 'OR' OR '&P5' EQ 'AND' OR '&P5' EQ 'ORIF' OR  +
               '&P5' EQ 'ANDIF').TWOPER2
         PUSHINS     (&P1,&P2,&P3,&P4,&P5,&P6)
&PF_CTR  SETA  &PF_CTR+4
         MEXIT
.TWOPER2 PUSHINS     (&P1,&P2,&P3,&P4,,&P6)
&PF_CTR  SETA  &PF_CTR+3
         MEXIT
.SGLOPR  GETCC  &P1(1)
         MEND
*********************************************************************
         MACRO
         PUSHINS     &PAM
         COPY PPFGBLCØ
         LCLA &WK,&I,&J,&K
&I       SETA  3
&J       SETA  4
&K       SETA  4
         AIF   ('&PAM(1)'(1,1) EQ 'B' OR '&PAM(1)' EQ 'EQU').BCH
         AIF   ('&PAM(5)' EQ '').TWOPERS
         AIF   ('&PAM(1)' EQ 'CS').CNSWAP
         AIF   ('&PAM(1)' EQ 'CDS').CNSWAP
         AIF   ('&PAM(1)'(1,1) EQ 'C').SETK
.CNSWAP  ANOP
&J       SETA  5
         AGO   .GETCOND
.TWOPERS AIF   ('&PAM(1)'(1,1) NE 'C').TSTIAC
         AIF   ('&PAM(1)' EQ 'CLCL').CLCL
&I       SETA  4
&J       SETA  3
         AGO   .SETK
.CLCL    ANOP
&I       SETA  3
&J       SETA  4
```

```
&K       SETA  3
         AGO   .SETK
.TSTIAC  ANOP
         AIF   ('&PAM(1)' NE 'IAC').SETK
&I       SETA  4
&J       SETA  3
&K       SETA  4
         AGO   .GETCOND
.SETK    ANOP
&K       SETA  5
.GETCOND GETCC &PAM(&J)
.BCH     AIF   (&PF_II GE 1ØØ).OVERI
&PF_II   SETA  &PF_II+1
&PF_IIND1(&PF_II)    SETC  '&PAM(1)'
&PF_IIND2(&PF_II)    SETC  '&PAM(2)'
         AIF   ('&PAM(&I)' NE '').LD31
&PF_IIND3(&PF_II)    SETC  ''
         AGO   .PAM4
.LD31    ANOP
&PF_IIND3(&PF_II)   SETC  '&PAM(&I)'
.PAM4    ANOP
         AIF   ('&PAM(&K)' NE '').LD41
&PF_IIND4(&PF_II)   SETC  ''
         AGO   .PAM5
.LD41    ANOP
&PF_IIND4(&PF_II)   SETC  '&PAM(&K)'
.PAM5    AIF   ('&PAM(6)' EQ '').BLKOUT5
         AIF   ('&PAM(6)'(1,1Ø) NE 'PF_C14LBL_').BLKOUT5
&PF_IIND5(&PF_II)   SETC  '&PAM(6)'
         MEXIT
.BLKOUT5 ANOP
&PF_IIND5(&PF_II)   SETC  ''
         MEXIT
.OVERI   MNOTE 8,'INSTRN STK SIZE EXCEEDED. FURTHER EXPANSIONS INVALID'
         MEND
*********************************************************************
         MACRO
         PUSHNEST &P1
         COPY  PPFGBLCØ
&PF_NI   SETA  &PF_NI+1
         AIF   (&PF_NI GE 5Ø).OVER
&PF_NEST(&PF_NI)     SETC  '    '.'&P1'
         MEXIT
.OVER    MNOTE 8,'NEST STACK SIZE EXCEEDED. FURTHER EXPANSIONS INVALID'
         MEND
*********************************************************************
         MACRO
         PUSHLAB
         COPY  PPFGBLCØ
         AIF   (&PF_LI GE 1ØØ).OVER
```

```
&PF_SEQ  SETA  &PF_SEQ+1
&PF_LI   SETA  &PF_LI+1
&PF_LIND(&PF_LI)     SETC  'PF_C14LBL_&PF_SEQ'
         MEXIT
.OVER    MNOTE 8,' LABEL STK SIZE EXCEEDED. FURTHER EXPANSIONS INVALID'
         MEND
*********************************************************************
         MACRO
         IFPROC
         COPY  PPFGBLCØ
         LCLB  &ANDIND,&ORIND
         PUSHLAB
&PF_CTR  SETA  2
&PF_ST(&PF_NI+1)     SETA  &PF_II+1
&PF_NEST(&PF_NI)     SETC  '  R'.'&PF_NEST(&PF_NI)'(4,5)
         AIF   (T'&SYSLIST(1) EQ 'O').LOOP
         AIF   (&SYSLIST(1) LE Ø OR &SYSLIST(1) GE 15).INVALCC
&PF_CCVAL      SETA  &SYSLIST(1)
         AIF   ('&SYSLIST(2)' EQ '').ENDBOOL
         MNOTE 4,'CC KEYWORD USED. OTHER PARAMETERS IGNORED'
         AGO   .ENDBOOL
.INVALCC MNOTE 4,'CC OUTSIDE VALID RANGE OF 1 TO 14. NOP GENERATED'
&PF_CCVAL SETA 15
         AGO   .ENDBOOL
.LOOP    STKINS    &SYSLIST(&PF_CTR),                                 +
               &SYSLIST(&PF_CTR+1),                                   +
               &SYSLIST(&PF_CTR+2),                                   +
               &SYSLIST(&PF_CTR+3),                                   +
               &SYSLIST(&PF_CTR+4)
         AIF   ('&SYSLIST(&PF_CTR+1)' EQ 'AND').ANDPROC
         AIF   ('&SYSLIST(&PF_CTR+1)' NE 'ANDIF').TESTOR
.ANDPROC PUSHINS   (BC,15-&PF_CCVAL,&PF_LIND(&PF_LI-1))
&ANDIND  SETB 1
         AIF   ('&SYSLIST(&PF_CTR+1)' NE 'ANDIF' OR NOT &ORIND).TESTLP
         POPINS    &PF_ST(&PF_NI+1)
&PF_LIND(&PF_LI)   EQU   *
&ORIND   SETB  Ø
&PF_LI   SETA  &PF_LI-1
         PUSHLAB
         AGO   .TESTLP
.TESTOR  AIF   ('&SYSLIST(&PF_CTR+1)' EQ 'OR').ORPROC
         AIF   ('&SYSLIST(&PF_CTR+1)' NE 'ORIF').TESTLP
.ORPROC  PUSHINS   (BC,&PF_CCVAL,&PF_LIND(&PF_LI))
&ORIND   SETB  1
         AIF   ('&SYSLIST(&PF_CTR+1)' NE 'ORIF' OR NOT &ANDIND).TESTLP
         PUSHINS (EQU,*,,,,&PF_LIND(&PF_LI-1))
&ANDIND  SETB Ø
         PUSHLAB
&PF_LI   SETA &PF_LI-1
&PF_LIND(&PF_LI-1) SETC  '&PF_LIND(&PF_LI+1)'
```

```
.TESTLP  ANOP
&PF_CTR  SETA  &PF_CTR+2
         AIF   ('&SYSLIST(&PF_CTR-1)' NE '').LOOP
.ENDBOOL AIF   ('&PF_NEST(&PF_NI)'(5,4) EQ 'DO').DOEND
         POPINS    &PF_ST(&PF_NI+1)
         BC    15-&PF_CCVAL,&PF_LIND(&PF_LI-1)
         AIF   (NOT &ORIND).POPLBL
&PF_LIND(&PF_LI)   EQU   *
.POPLBL  ANOP
&PF_LI   SETA  &PF_LI-1
         MEXIT
.DOEND   ANOP
&PF_CTR  SETA  &PF_ST(&PF_NI+1)
         AGO   .ENDLBL
.NXTLBL  AIF   ('&PF_IIND3(&PF_CTR)' NE '&PF_LIND(&PF_LI)').INCTR
&PF_IIND3(&PF_CTR) SETC  '&PF_LIND(&PF_LI-3)'
.INCTR   ANOP
&PF_CTR  SETA  &PF_CTR+1
.ENDLBL  AIF   (&PF_CTR LE &PF_II).NXTLBL
         POPINS    &PF_ST(&PF_NI+1)
         BC    &PF_CCVAL,&PF_LIND(&PF_LI-3)
         AIF   (NOT &ANDIND).POP2LBL                          @BA4340Ø5
&PF_LIND(&PF_LI-1) EQU   *
.POP2LBL ANOP
&PF_LI   SETA  &PF_LI-2
&PF_NEST(&PF_NI)   SETC  '   Y'.'&PF_NEST(&PF_NI)'(5,4)
         MEND
*********************************************************************
         MACRO
         IF    &P1,&P2,&P3,&P4,&P5,&P6,&P7,&P8,&P9,&PlØ,&P11,&P12,&P13,X
               &P14,&P15,&P16,&P17,&P18,&P19,&P2Ø,&P21,&P22,&P23,&P24, X
               &P25,&P26,&P27,&P28,&P29,&P3Ø,&P31,&P32,&P33,&P34,&P35, X
               &P36,&P37,&P38,&P39,&P4Ø,&P41,&P42,&P43,&P44,&P45,&P46, X
               &P47,&P48,&P49,&P5Ø,&CC=
         PUSHNEST IF
         PUSHLAB
         IFPROC    &CC,&P1,&P2,&P3,&P4,&P5,&P6,&P7,&P8,&P9,&PlØ,&P11,  X
               &P12,&P13,&P14,&P15,&P16,&P17,&P18,&P19,&P2Ø,&P21,&P22, X
               &P23,&P24,&P25,&P26,&P27,&P28,&P29,&P3Ø,&P31,&P32,&P33, X
               &P34,&P35,&P36,&P37,&P38,&P39,&P4Ø,&P41,&P42,&P43,&P44, X
               &P45,&P46,&P47,&P48,&P49,&P5Ø
         MEND
*********************************************************************
         MACRO
         ELSE  &OPTN                                           @BIAH3WI
         COPY  PPFGBLCØ
         AIF   ('&OPTN' EQ 'NULL').EXIT                        @BIAH3WI
&PF_LIND(&PF_LI+1) SETC  '&PF_LIND(&PF_LI)'
&PF_LI   SETA &PF_LI-1
         PUSHLAB
```

```
         AIF   ('&OPTN' EQ 'NOBRANCH').BYPBR                    @BIAH3WI
         BC    15,&PF_LIND(&PF_LI)
.BYPBR   ANOP                                                   @BIAH3WI
&PF_LIND(&PF_LI+1) EQU   *
.EXIT    ANOP                                                   @BIAH3WI
         MEND
*********************************************************************
         MACRO
         ENDIF
         COPY  PPFGBLCØ
         POPNEST   IF
&PF_LIND(&PF_LI)   EQU   *
&PF_LI   SETA  &PF_LI-1
         MEND
*********************************************************************
         MACRO
         DOPROC &FROM,&TO,&BY,&UNTIL,&WHILE,&P1
         COPY  PPFGBLCØ
         LCLA  &I
         LCLC  &LCLWK1
         PUSHLAB
         PUSHINS   (EQU,*,,,,&PF_LIND(&PF_LI))
&PF_ST(&PF_NI)     SETA  &PF_II+1
         PUSHLAB
         AIF   (T'&FROM EQ 'O').NOIND
         AIF   ('&FROM(3)' EQ '').INCR
         LA    &FROM(3),&PF_LIND(&PF_LI)
.INCR    ANOP
&I       SETA  &I+1
         AIF   ('&SYSLIST(&I,2)' EQ '').TEST
         AIF   ('&SYSLIST(&I,2)' EQ 'Ø').GENSR
         AIF   ('&SYSLIST(&I,2)'(1,1) EQ '-').NEGVAL
         AIF   (T'&SYSLIST(&I,2) EQ 'N').POSVAL
         AIF   ('&SYSLIST(&I,2)'(1,1) EQ '(').GENLR
         L     &SYSLIST(&I,1),&SYSLIST(&I,2)
         AGO   .TEST
.GENLR   LR    &SYSLIST(&I,1),&SYSLIST(&I,2)
         AGO   .TEST
.POSVAL  AIF   (&SYSLIST(&I,2) GE 4Ø96).TSTMAG
         LA    &SYSLIST(&I,1),&SYSLIST(&I,2)
         AGO   .TEST
.TSTMAG  AIF   (&SYSLIST(&I,2) GE 32768).FULLIT
         AGO   .HALFLIT
.NEGVAL  ANOP
&LCLWK1  SETC  '&SYSLIST(&I,2)'(2,7)
         AIF   (&LCLWK1 GE 32768).FULLIT
.HALFLIT LH    &SYSLIST(&I,1),=H'&SYSLIST(&I,2)'
         AGO   .TEST
.FULLIT  L     &SYSLIST(&I,1),=F'&SYSLIST(&I,2)'
         AGO   .TEST
```

```
.GENSR   SR    &SYSLIST(&I,1),&SYSLIST(&I,1)
.TEST    AIF   (&I LT 3).INCR
         AIF   (T'&UNTIL NE 'O').ERRMG2
.CKWHILE AIF   (T'&WHILE NE 'O').COMPGEN
&PF_LIND(&PF_LI)   EQU   *
.POSTIND AIF   (T'&P1 EQ 'O').GETIND
         AIF   (T'&BY NE 'O').PFB
         AIF   (T'&TO NE 'O').PFT
         AIF   ('&FROM(3)' NE '').BCTRZ
         PUSHINS   (BCT,&FROM(1),&PF_LIND(&PF_LI))
         AGO   .ERRMG
.BCTRZ   PUSHINS   (BCTR,&FROM(1),&FROM(3))
         AGO   .ERRMG
.PFT     PUSHINS   (&P1,&FROM(1),&TO(1),&PF_LIND(&PF_LI))
         MEXIT
         PUSHINS   (&P1,&FROM(1),&BY(1),&PF_LIND(&PF_LI))
         MEXIT
.GETIND  AIF   ('&FROM(3)' EQ '').BCTR1
         PUSHINS   (BCTR,&FROM(1),&FROM(3))
         MEXIT
.BCTR1   AIF   (T'&BY NE 'O').FB
         AIF   (T'&TO EQ 'O').FONLY
         PUSHINS   (BXLE,&FROM(1),&TO(1),&PF_LIND(&PF_LI))
         MEXIT
.FONLY   PUSHINS   (BCT,&FROM(1),&PF_LIND(&PF_LI))
         MEXIT
.FB      AIF   (T'&TO NE 'O').FTB
         AIF   ('&BY(2)' EQ '').GENBXLE
         AIF   ('&BY(2)'(1,1) NE '-').GENBXLE
         AGO   .GENBXH
.FTB     AIF   ('&TO(2)' EQ '' OR '&FROM(2)' EQ '').GENBXLE
         AIF   ('&FROM(2)'(1,1) EQ '-').TRYTNEG
         AIF   (T'&FROM(2) NE 'N').GENBXLE
         AIF   ('&TO(2)'(1,1) EQ '-').GENBXH
         AIF   (T'&TO(2) NE 'N').GENBXLE
         AIF   (&FROM(2) GT &TO(2)).GENBXH
.GENBXLE PUSHINS   (BXLE,&FROM(1),&BY(1),&PF_LIND(&PF_LI))
         MEXIT
.TRYTNEG AIF   ('&TO(2)'(1,1) NE '-').GENBXLE
         AIF   ('&FROM(2)'(2,7) GE '&TO(2)'(2,7)).GENBXLE
.GENBXH  PUSHINS   (BXH,&FROM(1),&BY(1),&PF_LIND(&PF_LI))
         MEXIT
.NOIND   AIF   (T'&WHILE EQ 'O').NOWHILE
         AIF   (T'&UNTIL NE 'O').COMPGEN
         BC    15,&PF_LIND(&PF_LI)
         PUSHLAB
&PF_LI   SETA  &PF_LI-1
&PF_LIND(&PF_LI+1) EQU *
         AIF   ('&WHILE(6)' EQ '').OKSUBL
         STKINS    &WHILE
```

```
         MEXIT
.OKSUBL  STKINS    (&WHILE(1),&WHILE(2),&WHILE(3),&WHILE(4),           X
               &WHILE(5),&PF_LIND(&PF_LI))
         AIF   ('&WHILE(2)' EQ '').LABEL
         PUSHINS   (BC,&PF_CCVAL,&PF_LIND(&PF_LI+1))
         MEXIT
.LABEL   PUSHINS   (BC,&PF_CCVAL,&PF_LIND(&PF_LI+1),,,&PF_LIND(&PF_LI))
         MEXIT
.NOWHILE AIF   (T'&UNTIL EQ 'O').TRYINF
&PF_LIND(&PF_LI)   EQU   *
.UNT     STKINS    &UNTIL
         PUSHINS   (BC,15-&PF_CCVAL,&PF_LIND(&PF_LI))
         MEXIT
.TRYINF  AIF   ('&P1' NE 'INF').ERRMG1
&PF_LIND(&PF_LI)   EQU   *
         PUSHINS   (BC,15,&PF_LIND(&PF_LI))
         MEXIT
.COMPGEN AIF   ('&WHILE(6)' EQ '').OK
         STKINS    &WHILE
         AGO   .BCHINST
.OK      STKINS    (&WHILE(1),&WHILE(2),&WHILE(3),&WHILE(4),           X
               &WHILE(5),&PF_LIND(&PF_LI))
         AIF   (N'&WHILE GT 1).ENDCOMP
&PF_LIND(&PF_LI)   BC    15-&PF_CCVAL,&PF_LIND(&PF_LI-1)
         AGO   .FLAGEQU
.ENDCOMP ANOP
&PF_ST(&PF_NI+1)   SETA  &PF_II
         POPINS    &PF_ST(&PF_NI+1)
.BCHINST BC    15-&PF_CCVAL,&PF_LIND(&PF_LI-1)
.FLAGEQU ANOP
&PF_NEST(&PF_NI)   SETC  '   Y'.'&PF_NEST(&PF_NI)'(5,4)
         AIF   (T'&FROM NE 'O').POSTIND
         AGO   .UNT
.ERRMG   MNOTE 4,'POSITIONAL PARAMETER IGNORED. BCT/BCTR LOOP END USED'
         MEXIT
.ERRMG2  MNOTE 4,'UNTIL KEYWORD INVALID WITH INDEXING GROUP. IGNORED'
         AGO   .CKWHILE
.ERRMG1  MNOTE 4,'NO WHILE,UNTIL,OR INDEXING PARAMETERS ON DO MACRO.'
         MEND
*********************************************************************
         MACRO
         DO    &P1,&FROM=,&TO=,&BY=,&UNTIL=,&WHILE=
         PUSHNEST DO
         DOPROC    &FROM,&TO,&BY,&UNTIL,&WHILE,&P1
         MEND
*********************************************************************
         MACRO
         DOEXIT    &P1,&P2,&P3,&P4,&P5,&P6,&P7,&P8,&P9,&P1Ø,&P11,&P12, X
               &P13,&P14,&P15,&P16,&P17,&P18,&P19,&P2Ø,&P21,&P22,&P23, X
               &P24,&P25,&P26,&P27,&P28,&P29,&P3Ø,&P31,&P32,&P33,&P34, X
```

```
              &P35,&P36,&P37,&P38,&P39,&P4Ø,&P41,&P42,&P43,&P44,&P45, X
              &P46,&P47,&P48,&P49,&P5Ø,&CC=
         COPY  PPFGBLCØ
         PUSHLAB
&PF_NEST(&PF_NI)   SETC  '   Y'.'&PF_NEST(&PF_NI)'(5,4)
         IFPROC    &CC,&P1,&P2,&P3,&P4,&P5,&P6,&P7,&P8,&P9,&P1Ø,&P11,  X
              &P12,&P13,&P14,&P15,&P16,&P17,&P18,&P19,&P2Ø,&P21,&P22, X
              &P23,&P24,&P25,&P26,&P27,&P28,&P29,&P3Ø,&P31,&P32,&P33, X
              &P34,&P35,&P36,&P37,&P38,&P39,&P4Ø,&P41,&P42,&P43,&P44, X
              &P45,&P46,&P47,&P48,&P49,&P5Ø
         MEND
*********************************************************************
         MACRO
         ENDDO
         GBLA   &PF_ST(51),&PF_NI,&PF_LI,&PF_II
         POPINS &PF_ST(&PF_NI)
&PF_II   SETA   &PF_II-1
         POPNEST   DO
&PF_LI   SETA  &PF_LI-2
         MEND
*********************************************************************
         MACRO
         STRTSRCH  &P1,&FROM=,&TO=,&BY=,&UNTIL=,&WHILE=
         PUSHLAB
         PUSHNEST  SRCH
         DOPROC    &FROM,&TO,&BY,&UNTIL,&WHILE,&P1
         PUSHLAB
         MEND
*********************************************************************
         MACRO
         EXITIF    &P1,&P2,&P3,&P4,&P5,&P6,&P7,&P8,&P9,&P1Ø,&P11,&P12, X
               &P13,&P14,&P15,&P16,&P17,&P18,&P19,&P2Ø,&P21,&P22,&P23, X
               &P24,&P25,&P26,&P27,&P28,&P29,&P3Ø,&P31,&P32,&P33,&P34, X
               &P35,&P36,&P37,&P38,&P39,&P4Ø,&P41,&P42,&P43,&P44,&P45, X
               &P46,&P47,&P48,&P49,&P5Ø,&CC=
         IFPROC    &CC,&P1,&P2,&P3,&P4,&P5,&P6,&P7,&P8,&P9,&P1Ø,&P11,  X
               &P12,&P13,&P14,&P15,&P16,&P17,&P18,&P19,&P2Ø,&P21,&P22, X
               &P23,&P24,&P25,&P26,&P27,&P28,&P29,&P3Ø,&P31,&P32,&P33, X
               &P34,&P35,&P36,&P37,&P38,&P39,&P4Ø,&P41,&P42,&P43,&P44, X
               &P45,&P46,&P47,&P48,&P49,&P5Ø
         MEND
*********************************************************************
         MACRO
         ORELSE
         COPY  PPFGBLCØ
&PF_LIND(&PF_LI+1) SETC  '&PF_LIND(&PF_LI)'
&PF_LI   SETA  &PF_LI-1
         PUSHLAB
         BC    15,&PF_LIND(&PF_LI-3)
&PF_LIND(&PF_LI+1) EQU   *
```

```
&PF_NEST(&PF_NI)   SETC  '  P'.'&PF_NEST(&PF_NI)'(4,5)
         MEND
*********************************************************************
         MACRO
         ENDLOOP
         COPY  PPFGBLCØ
         AIF   ('&PF_NEST(&PF_NI)'(3,1) EQ 'P').CALLEND
         BC    15,&PF_LIND(&PF_LI-3)
&PF_LIND(&PF_LI)   EQU   *
.CALLEND ANOP
&PF_NEST(&PF_NI)   SETC  '   '.'&PF_NEST(&PF_NI)'(4,5)
         POPINS    &PF_ST(&PF_NI)
&PF_II   SETA  &PF_II-1
&PF_LI   SETA  &PF_LI-3
         MEND
*********************************************************************
         MACRO
         ENDSRCH
         COPY  PPFGBLCØ
         POPNEST   SRCH
&PF_LIND(&PF_LI)   EQU   *
&PF_LI   SETA  &PF_LI-1
         MEND
*********************************************************************
         MACRO
         CASENTRY  &P1,&VECTOR=,&POWER=Ø
         COPY  PPFGBLCØ
         PUSHNEST  CASE
         PUSHLAB
         PUSHLAB
         AIF   (&PF_AI GE 5Ø).OVER
&PF_AI   SETA  &PF_AI+1
&PF_AIND(&PF_AI)   SETA  Ø
&PF_RIND(&PF_AI)   SETC  '&P1'
&PF_MULT(&PF_AI)   SETA  1
&PF_CTR  SETA &POWER
.SHIFTLP AIF   (&PF_CTR LE Ø).GENSHFT
&PF_MULT(&PF_AI)   SETA  &PF_MULT(&PF_AI)+&PF_MULT(&PF_AI)
&PF_CTR  SETA  &PF_CTR-1
         AGO   .SHIFTLP
.GENSHFT AIF   (&PF_MULT(&PF_AI) EQ 4).TESTVEC
         AIF   (&PF_MULT(&PF_AI) GT 4).RTSHIFT
         SLA   &P1,2-&POWER
         AGO   .TESTVEC
.RTSHIFT SRA   &P1,&POWER-2
.TESTVEC AIF   ('&VECTOR' EQ 'B' OR '&VECTOR' EQ 'BR').BRVEC
         PUSHLAB
         A     &P1,&PF_LIND(&PF_LI)
         L     &P1,Ø(&P1)
         BR    &P1
```

```
&PF_LIND(&PF_LI)   DC    A(&PF_LIND(&PF_LI-2))
&PF_LI   SETA  &PF_LI-1
         MEXIT
.BRVEC   BC    15,&PF_LIND(&PF_LI-1)(&P1)
&PF_NEST(&PF_NI)   SETC  '   B'.'&PF_NEST(&PF_NI)'(5,4)
         MEXIT
.OVER    MNOTE 8,'TOTAL CASES STK EXCEEDED. FURTHER EXPANSIONS INVALID'
         MEND
*********************************************************************
         MACRO
         CASE
         COPY  PPFGBLCØ
         LCLA  &NBR,&CASENO
         PUSHLAB
         AIF   (N'&SYSLIST EQ 1).LDSUBL
&NBR     SETA  N'&SYSLIST
         AGO   .LDAIND
.LDSUBL  ANOP
&NBR     SETA  N'&SYSLIST(1)
.LDAIND  AIF   (&NBR LE Ø).NOPRMS
&PF_AIND(&PF_AI)   SETA  &PF_AIND(&PF_AI)+&NBR
.TSTSUBL AIF   (T'&SYSLIST(1,2) EQ 'O' AND &NBR NE 1).NOTSUBL
&CASENO  SETA  &SYSLIST(1,&NBR)
         AGO  .TSTMULT
.NOTSUBL ANOP
&CASENO  SETA  &SYSLIST(&NBR)
.TSTMULT AIF   (&CASENO-(&CASENO/&PF_MULT(&PF_AI))*&PF_MULT(&PF_AI) NE +
               Ø).NOTMULT
         AIF   (&CASENO EQ Ø).NOTMULT
         AIF   (&PF_CI GE 2ØØ).OVER
&PF_CI   SETA  &PF_CI+1
&PF_CIND1(&PF_CI)  SETA  &CASENO
&PF_CIND2(&PF_CI)  SETC  '&PF_LIND(&PF_LI)'
.RETRNPT ANOP
&NBR     SETA  &NBR-1
         AIF   (&NBR NE Ø).TSTSUBL
.FRSTIME AIF   ('&PF_NEST(&PF_NI)'(3,1) NE ' ').BCGEN1
&PF_NEST(&PF_NI)   SETC  '  Y'.'&PF_NEST(&PF_NI)'(4,5)
         AGO   .EQUGN1
.BCGEN1  AIF   ('&PF_NEST(&PF_NI)'(4,1) EQ 'B').BCINST
         L     &PF_RIND(&PF_AI),&PF_LIND(&PF_LI-2)
         BR    &PF_RIND(&PF_AI)
         AGO   .EQUGN1
.BCINST  B     &PF_LIND(&PF_LI-1)
.EQUGN1  ANOP
&PF_LIND(&PF_LI)   EQU   *
&PF_LI   SETA  &PF_LI-1
         MEXIT
.NOTMULT MNOTE 8,'CASE &CASENO DELETED. NOT MULTIPLE OF &PF_MULT(&PF_AI+
               ).'
```

```
&PF_AIND(&PF_AI)   SETA  &PF_AIND(&PF_AI)-1
         AGO   .RETRNPT
.NOPRMS  MNOTE 'NO PARAMETERS FOUND WITH CASE MACRO'
         AGO   .FRSTIME
.OVER    MNOTE 8,'CASE NUMBER STK EXCEEDED. FURTHER EXPANSIONS INVALID'
         MEND
*********************************************************************
         MACRO
         ENDCASE
         COPY  PPFGBLCØ
         ACTR  99999
         LCLA  &K,&I
         AIF   ('&PF_NEST(&PF_NI)'(4,1) EQ 'B').BVECT1
         L     &PF_RIND(&PF_AI),&PF_LIND(&PF_LI-1)
         BR    &PF_RIND(&PF_AI)
&PF_LIND(&PF_LI-1) DC    A(&PF_LIND(&PF_LI))
         AGO   .BLDVECT
.BVECT1  ANOP
&PF_LIND(&PF_LI-1) B     &PF_LIND(&PF_LI)
.BLDVECT AIF   (&PF_AIND(&PF_AI) LE Ø).TESTCI
&K       SETA  &PF_MULT(&PF_AI)
.LOOPIN  ANOP
&I       SETA  1
.LOOP1   AIF   (&K EQ &PF_CIND1(&PF_CI-&I+1)).ELEND
         AIF   (&I EQ &PF_AIND(&PF_AI)).GENTRY
&I       SETA  &I+1
         AGO   .LOOP1
.GENTRY  AIF   ('&PF_NEST(&PF_NI)'(4,1) EQ 'B').BVECT2
         DC    A(&PF_LIND(&PF_LI))
         AGO   .INCRK
.ELEND   AIF   ('&PF_NEST(&PF_NI)'(4,1) EQ 'B').BVECT3
         DC    A(&PF_CIND2(&PF_CI-&I+1))
         AGO   .DECSTK
.BVECT3  B     &PF_CIND2(&PF_CI-&I+1)
.DECSTK  ANOP
&PF_AIND(&PF_AI)   SETA  &PF_AIND(&PF_AI)-1
&PF_CI   SETA  &PF_CI-1
         AIF   (&PF_AIND(&PF_AI) EQ Ø).TESTCI
.LOOP2   AIF   (&I EQ 1).INCRK
&I       SETA &I-1
&PF_CIND1(&PF_CI-&I+1)   SETA  &PF_CIND1(&PF_CI-&I+2)
&PF_CIND2(&PF_CI-&I+1)   SETC  '&PF_CIND2(&PF_CI-&I+2)'
         AGO   .LOOP2
.BVECT2  B     &PF_LIND(&PF_LI)
.INCRK   ANOP
&K       SETA  &K+&PF_MULT(&PF_AI)
         AGO   .LOOPIN
.TESTCI  AIF   (&PF_CI LT Ø).ASTKERR
&PF_LIND(&PF_LI)   EQU   *
&PF_LI   SETA  &PF_LI-2
```

```
&PF_AI   SETA  &PF_AI-1
         POPNEST   CASE
         AIF   (&PF_AI LT Ø).ASTKERR
         MEXIT
.ASTKERR MNOTE 8,'NEGATIVE CASE MACRO STACK PTR. EXPANSION INVALID.'
         MEND
*********************************************************************
         MACRO
         SELECT    &EVERY
         COPY  PPFGBLCØ
         GBLC  &PF_ESEL(5Ø)
         GBLB  &PF_EVRY(5Ø)
         PUSHNEST  SEL
&PF_EVRY(&PF_NI)   SETB  ('&EVERY' EQ 'EVERY')
&PF_ESEL(&PF_NI)   SETC  ''
         MEND
*********************************************************************
         MACRO
         WHEN  &P1,&P2,&P3,&P4,&P5,&P6,&P7,&P8,&P9,&P1Ø,&P11,&P12,&P13,X
               &P14,&P15,&P16,&P17,&P18,&P19,&P2Ø,&P21,&P22,&P23,&P24, X
               &P25,&P26,&P27,&P28,&P29,&P3Ø,&P31,&P32,&P33,&P34,&P35, X
               &P36,&P37,&P38,&P39,&P4Ø,&P41,&P42,&P43,&P44,&P45,&P46, X
               &P47,&P48,&P49,&P5Ø,&CC=
         COPY  PPFGBLCØ
         GBLC  &PF_ESEL(5Ø)
         GBLB  &PF_EVRY(5Ø)
         GBLB  &PF_NONSELD(5Ø)
         AIF   ('&PF_ESEL(&PF_NI)' NE '').TSTEVRY
         AIF   ('&P1' EQ 'NONE').NONE1ST
         PUSHLAB
&PF_ESEL(&PF_NI)   SETC  '&PF_LIND(&PF_LI)'
         AGO   .BYPASS
.TSTEVRY AIF   (&PF_EVRY(&PF_NI)).TSTNON
         B     &PF_ESEL(&PF_NI)
         AGO   .CONT
.TSTNON  AIF   ('&P1' EQ 'NONE').BADEVRY
.CONT    ANOP
&PF_LIND(&PF_LI)   EQU   *
&PF_LI   SETA  &PF_LI-1
         AIF   (&PF_NONSELD(&PF_NI)).BADWHEN
.BYPASS  ANOP
&PF_NONSELD(&PF_NI)      SETB  ('&P1' EQ 'NONE')
         AIF   ('&P1' NE 'NONE').DOIF
         AIF   (&PF_EVRY(&PF_NI)).BADEVRY
         PUSHNEST  IF
         PUSHLAB
         AGO   .EXIT
.DOIF    PUSHNEST  IF
         PUSHLAB
         IFPROC    &CC,&P1,&P2,&P3,&P4,&P5,&P6,&P7,&P8,&P9,&P1Ø,&P11,  X
```

```
              &P12,&P13,&P14,&P15,&P16,&P17,&P18,&P19,&P2Ø,&P21,&P22, X
              &P23,&P24,&P25,&P26,&P27,&P28,&P29,&P3Ø,&P31,&P32,&P33, X
              &P34,&P35,&P36,&P37,&P38,&P39,&P4Ø,&P41,&P42,&P43,&P44, X
              &P45,&P46,&P47,&P48,&P49,&P5Ø
.EXIT    POPNEST   IF
         MEXIT
.NONE1ST MNOTE 8,'''NONE'' INVALID IN THE FIRST WHEN OF A SELECT STRUCT+
              URE'
         MEXIT
.BADEVRY MNOTE 8,'''NONE'' OPTION INVALID WITH ''SELECT EVERY'''
         MEXIT
.BADWHEN MNOTE 8,'NO WHEN STATEMENT ALLOWED AFTER ''WHEN NONE'''
         MEND
*********************************************************************
         MACRO
         ENDSEL
         COPY  PPFGBLCØ
         GBLC  &PF_ESEL(5Ø)
         GBLB  &PF_EVRY(5Ø)
         GBLB  &PF_NONSELD(5Ø)
         AIF   (&PF_EVRY(&PF_NI)).ISEVRY
&PF_ESEL(&PF_NI)   EQU   *
         AIF   (&PF_NONSELD(&PF_NI)).DONE
.ISEVRY  ANOP
&PF_LIND(&PF_LI)   EQU   *
.DONE    POPNEST   SEL
&PF_LI   SETA  &PF_LI-2
         MEND
```

PPFGBLC0

```
         GBLA  &PF_CCVAL              COND CODE VARIABLE
         GBLA  &PF_CTR                MACRO PARAMETER COUNTER
         GBLA  &PF_SEQ                LABEL NUMBER GENERATOR
         GBLA  &PF_AI                 INDEX FOR TOTAL NO. CASES STK
         GBLA  &PF_CI                 INDEX FOR CASE AND LBL NO. STKS
         GBLA  &PF_II                 PTR TO INST STKS
         GBLA  &PF_LI                 INDEX FOR LABEL NUMBER STK
         GBLA  &PF_NI                 PTR TO NEST STK
         GBLA  &PF_AIND(5Ø)           TOTAL CASES STK
         GBLA  &PF_CIND1(2ØØ)         CASE NUMBER STK
         GBLA  &PF_MULT(5Ø)           CASE NUMBER MULTIPLIER
         GBLA  &PF_ST(51)             INST STK INCREASE AT EACH LEVEL
         GBLC  &PF_CIND2(2ØØ)         LABEL NUMBER STK FOR CASES
         GBLC  &PF_IIND1(1ØØ)         INSTRUCTION STK 1
         GBLC  &PF_IIND2(1ØØ)         INSTRUCTION STK 2
.*       GBLC  &PF_I22(1ØØ)           INSTRUCTION STK 2, 2ND PART
.*       GBLC  &PF_I23(1ØØ)           INSTRUCTION STK 2, 3RD PART
.*       GBLC  &PF_I24(1ØØ)           INSTRUCTION STK 2, 4TH PART
```

```
         GBLC  &PF_IIND3(100)          INSTRUCTION STK 3
.*       GBLC  &PF_I32(100)            INSTRUCTION STK 3, 2ND PART
.*       GBLC  &PF_I33(100)            INSTRUCTION STK 3, 3RD PART
.*       GBLC  &PF_I34(100)            INSTRUCTION STK 3, 4TH PART
         GBLC  &PF_IIND4(100)          INSTRUCTION STK 4
.*       GBLC  &PF_I42(100)            INSTRUCTION STK 4, 2ND PART
.*       GBLC  &PF_I43(100)            INSTRUCTION STK 4, 3RD PART
         GBLC  &PF_IIND5(100)          INSTRUCTION NAME STACK
         GBLC  &PF_LIND(101)           LABEL NUMBER STK
         GBLC  &PF_NEST(50)            NESTING STK
         GBLC  &PF_RIND(50)            REG STK FOR CASENTRY MACRO
```

Pieter Wiid
Systems Engineer
OutSource (South Africa)

Displaying multi-volume dataset details

INTRODUCTION

In our data centre we are using many multi-volume datasets because of some applications' huge data needs. So we frequently need to know the details of each multi-volume dataset – such as on how many volumes it is allocated, how many extents, and how many tracks/cylinders it uses on each volume.

To be able to see all volumes of a multi-volume dataset, normally the I (dataset information) command is issued in the dataset list panel that is created by the Dslist utility (ISPF Option 3.4) for non-VSAM datasets. The result panel will display the dataset, allocation volume, and a + sign to show that the dataset spans multiple volumes. The dataset allocation attributes are the same for all spanned volumes. If a + sign is displayed then the *Enter* key is pressed to display all the spanned volumes. All the volumes that span a multiple volume dataset that the system will allow to be displayed will be shown. The number of volumes allocated to this dataset will also be displayed. The same information can also be got by issuing the TSO LISTCAT command. However, when it comes to gathering extents distribution throughout all volumes, neither method can help us. The only way is to issue the LISTDSI command and

interpret its variables within a REXX program.

Allocation details of a VSM multi-volume dataset can be obtained by issuing the TSO LISTCAT command. It gives all the volume names of the dataset, space allocation, and extents distribution throughout all volumes.

To ease the manual procedures explained above, I came up with the REXX LISTMV, which extracts and displays all the details of multi-volume datasets on a dynamically-built panel in the REXX. This way, we can quickly take a look at all details of a multi-volume dataset without having to deal with overwhelming LISTCAT output. The REXX will be in charge of interpreting LISTCAT command output for VSAM datasets and LISTDSI command variables for non-VSAM datasets.

Displaying all the volume details of a multi-volume dataset on one or more successive panels can help us draw some important conclusions as well. For example, we can observe how many extents exist on a single disk volume and see whether the volume is too fragmented and therefore needs defragmenting.

HOW TO EXECUTE THE REXX LISTMV

First, the REXX has to be copied to one of your Sysproc or Sysexec libraries. Although this REXX uses one or more panels, you don't need to allocate any ISPPLIB libraries since the REXX will allocate a temporary panel library and build the necessary panels inside it. At the end, the REXX will free the panel library.

To execute it for a multi-volume dataset, the desired multi-volume dataset is displayed by using the Dslist utility (ISPF Option 3.4). **LISTMV** is written on the command line, then the *Enter* key is pressed. The REXX will use as many panels as necessary to display all the details of a multi-volume dataset. Note that each panel can have a maximum of 10 volume details, so details of a multi-volume dataset spanning 59 volumes will fit up to 6 panels. On the last panel, we will see the total space usage and all extent distributions. As for VSAM datasets, the exact amount of allocated space and used extents will be separately shown for each component.

There is no problem issuing the LISTMV command for single-volume datasets or any other type of dataset, such as PDSE datasets that can't be multi-volume. In this case, LISTMV will display a panel saying that it's not a multi-volume dataset and some characteristics, eg tape dataset, PO dataset, GDG dataset, on this panel.

SAMPLE EXECUTION

Sequential dataset:

```
           MULTI VOLUME DATA SET DETAILS

Data set: EX.ETI.MARTA.VILLAR.PERNAS
Single volume data set   PS   data set.
Volume  Alloc   Used   Primary   Second   Numext  Vseq
PET116  38500   38500     8500    10000        4  1
PET105  30000   30000    10000    10000        3  2
PET110  40000   40000    10000    10000        4  3
PET112  40000   40000    10000    10000        4  4
PET115  30000   22510    10000    10000        3  5

Space unit     :     TRACK  Total extents   :    18
Space allocated:    178500  Num. of volumes :     5
Space used     :    171010

Hit <Enter> to exit.
```

VSAM (KSDS) dataset:

```
           MULTI VOLUME DATA SET DETAILS

 Data set: EX.ETI.HSM.SIL.VSAM
  Multi-volume  data set   VSAM data set.
 Volume  Alloc   Comp  Primary   Second    Numext  Vseq
 PET113    123   DATA        1        1       123  1
 PET121    123   DATA        1        1       123  2
 PET104      5   DATA        1        1         5  3
 PET113      6  INDEX        1        1         6  4

 Space unit (D,I): TRK,TRK   SPACE USAGE    : (in tracks)
 Tot.extents(D,I):   251,6   Total alloc (D):        251
 Num.of vol.(D,I):     3,1   Total alloc (I):          6

 Hit <Enter> to exit.
```

VSAM (ESDS) dataset:

```
           MULTI VOLUME DATA SET DETAILS
```

```
Data set: DSNDB2E.DSNDBC.GROUPE01.OSMOTS32.I0001.A138
 Multi-volume  data set   VSAM data set.
Volume  Alloc   Comp  Primary   Second    Numext  Vseq
PDB114      2   DATA        2      139         1  1
PDB113   1390   DATA        2      139        10  2
PDB103   1529   DATA        2      139        11  3
PDB105   1807   DATA        2      139        13  4

Space unit (D,I):   CYL,-   SPACE USAGE    : (in tracks)
Tot.extents(D,I):    35,0   Total alloc (D):      70920
Num.of vol.(D,I):     4,0   Total alloc (I):          0

Hit <Enter> to exit.
```

WHAT'S THE MAXIMUM NUMBER OF VOLUMES FOR STORING A MULTI-VOLUME DATASET?

PDS and PDSE datasets are limited to one volume, and so can't be multi-volume datasets. Extended format sequential datasets with multiple stripes are limited to 16 volumes. A dataset on a VIO simulated device is limited to 65,535 tracks and is on one volume. A VSAM dataset can have 255 extents (123 extents per volume) combined over all volumes of a multi-volume VSAM dataset. All other DASD datasets are limited to 59 volumes. For this reason, the Data Class *Volume Count* field can't have a value greater than 59.

Tape datasets are limited to 255 volumes. (This is not the scope of the REXX LISTMV though.) HFS datasets can expand to as many as 255 extents of DASD space on multiple volumes (59 volumes maximum with 123 extents per volume).

HOW ARE MULTI-VOLUME DATASETS ALLOCATED?

ISPF Option 3.2 provides the ability to allocate non-SMS or SMS multiple volume datasets through the selection field. This field displays a panel that allows users to enter the number of volumes or the names of the volumes. For non-SMS datasets, if a number is entered, the names will be ignored. The allocation attributes entered on the allocation panel will span all volumes entered. The value entered in the volume serial field of the allocation panel will be the value that is placed in the first volume field of the multivolume allocation panel.

For SMS datasets, the names of volumes don't make sense since SMS will choose volumes. On the other hand, the number entered for the *number of volumes* field will override the volume count value defined in the dataset's data class. For example, if the volume count is 5 in the data class and we enter 2 in the allocation panel, then just two volumes will be allocated for the dataset we've just defined.

SOME EXAMPLES OF DEFINING MULTI-VOLUME DATA SETS

VSAM multi-volume dataset allocation:

```
DEF CLUSTER(NAME(x) VOLUME(*,*,*,*) CYLINDERS(4))
```

You can let SMS choose the volumes for SMS-managed datasets by coding an asterisk (*) for the volser with the VOLUMES parameter. For SMS-managed and non-SMS-managed datasets, you can specify up to 59 volume serial numbers.

Non-VSAM multi-volume dataset allocation:

```
ALLOC DA('x.y') LRECL(8Ø) BLKSIZE(Ø) NEW CATALOG RECFM(F) DSORG(DA)
CYLINDER SPACE(1,1) RELEASE VOLUIME(PEX1Ø1,PEX1Ø2,PEX1Ø3) MAXVOL(3)
```

If it was an SMS dataset, we wouldn't need to assign any volser numbers. In this case, the dataset would be allocated on volumes which SMS will decide. The following JCL requests that the system assign five volumes to the dataset:

```
//PERNAS   DD   DSN=INEXØØ4.MARTAPV.DENIZ,UNIT=(SYSALLDA,5),
// SPACE=(CYL,3,RLSE),DCB=(RECFM=FB,LRECL=129,BLKSIZE=Ø),DISP=(,CATLG)
```

The dataset in the following example will first allocate two volumes, serial numbers PEX101 and PEX102. The VOLUME *volume count* subparameter requests four volumes, if required. Thus, if more space is required, the system can assign a third and fourth volume:

```
//PERNAS   DD   DSN=INEXØØ4.MARTAPV.MAR,UNIT=SYSALLDA,
// SPACE=(CYL,3,RLSE),DCB=(RECFM=FB,LRECL=129,BLKSIZE=Ø),DISP=(,CATLG),
// VOLUME=(,,,4,SER=(PEX1Ø1,PEX1Ø2))
```

LISTMV SOURCE

```
/*REXX*/
/* Rexx     : Listmv                                                 */
```

```
/* Function : Displays multi-volume dataset details.                  */
Parse Arg Fi         /* Fi : Dataset name                             */
Status=Msg('Off')    /* Suppress TSO messages issued by the Listdsi.  */
Fi = Strip(Fi,B,"'") /* Remove both leading & leading characters.     */
Call Clear           /* Clear variable names and values from pool.    */
Call Volcnt          /* Find volume count for the dataset.            */
If Vsam=1 Then       /* Processing of VSAM datasets.                  */
       Call ProcVsam
          Else       /* Processing of non-VSAM datasets.     */
       Call ProcNVsam
"Free F("Ddname")"   /* Free the temporary panel library.             */
EXIT /*Main REXX */
CLEAR:
/* Initialize variables.                          */
TotA=Ø;TotE=Ø;TotU=Ø;Mig=Ø;TotE_D=Ø;TotE_I=Ø;Msg1="";Msg2=""
Blank= "                                          "
"Ispexec Verase (kØ k1  k2  k3  k4  k5  k6)   Profile"
"Ispexec Verase (k7  k8  k9  k1Ø k11 k12)  Profile"
"Ispexec Verase (k13 k14 k15 k16 k17 k18)  Profile"
"Ispexec Verase (k19 k2Ø k21 k22 k23 k24)  Profile"
"Ispexec Verase (k25 k26 k27 k28 k29 k3Ø)  Profile"
"Ispexec Verase (k31 k32 k33 k34 k35 k36)  Profile"
"Ispexec Verase (k37 k38 k39 k4Ø k41 k42)  Profile"
"Ispexec Verase (k43 k44 k45 k46 k47 k48)  Profile"
"Ispexec Verase (k49 k5Ø k51 k52 k53 k54)  Profile"
"Ispexec Verase (k55 k56 k57 k58 k59 k6Ø)  Profile"
"Ispexec Verase (Fi,y,TotE,TotA,TotU)      Profile"
RETURN /* End-of-the-procedure-CLEAR  */
VOLCNT:
Vsam   = Ø      /* Flag to determine if it's a VSAM dataset          */
Hfs    = Ø      /* Flag to determine if it's a HFS  dataset          */
Pdse   = Ø      /* Flag to determine if it's a PDSE dataset          */
Gdg    = Ø      /* Flag to determine if it's a GDG  dataset          */
Dat    = Ø      /* Flag to determine Vsam component type             */
Cnt2   = Ø      /* Flag to determine if a data set is multi-volume?  */
Cnt    = Ø      /* Total volume count                                */
Cnt_D  = Ø      /* Total volume count for Data  component of VSAM    */
Cnt_I  = Ø      /* Total volume count for Index component of VSAM    */
Sipa_D = Ø      /* Total space usage  for Data  component of VSAM    */
Sipa_I = Ø      /* Total space usage  for Index component of VSAM    */
Tot_AllocD. = Ø /* Stem for alloc'd space on each vol.of a Mv dataset */
Tot_AllocI. = Ø /* Stem for alloc'd space on each vol.of a Mv dataset */
Space_TypeD = '-';Space_TypeI = '-'
X = Outtrap('Martapv.')
"Listc Entry("Fi") All"                    /* Issue "Listc" command. */
X = Outtrap('Off')
If Index('CLUSTERDATA —INDEX -',Substr(Martapv.1,1,7)) <> Ø
                                 Then Vsam=1
If Index(Martapv.1,'GDG BASE') <> Ø Then Gdg=1
If Index(Martapv.7,'HFS')     <> Ø Then Hfs=1
```

```
If Index(Martapv.7,'LIBRARY') <> Ø Then Pdse=1
Do u = 1 to Martapv.Ø
If Vsam=1 Then
   Do
      If Index(Substr(MartapV.u,1,2Ø),'DATA ——') <> Ø Then Dat = 1
      If Index(Substr(MartapV.u,1,2Ø),'INDEX ——') <> Ø Then Dat = 2
   End
Volser=Substr(Martapv.u,26,6)
ju=Index(Martapv.u,'VOLSER—')
If  ju <> Ø  Then  Cnt2 = Cnt2 + 1
If (ju <> Ø & Volser <> '——*' & Dat = Ø) Then Cnt   = Cnt   + 1
If (ju <> Ø & Volser <> '——*' & Dat = 1) Then Cnt_D = Cnt_D + 1
If (ju <> Ø & Volser <> '——*' & Dat = 2) Then Cnt_I = Cnt_I + 1
If Index(Martapv.u,'LOW-CCHH') <> Ø Then
   Do
     ku=Index(MartapV.u,'TRACKS——')
     Sipa = Substr(MartapV.u,ku+6)     /* Extract the space amount.   */
     Sipa = Translate(Sipa,'','-')
     If Dat=1 Then Do
                    Sipa_D = Sipa_D + Sipa
                    If Space_Type=CYL Then Sipa = Sipa%15
                    Tot_AllocD.Cnt_D = Tot_AllocD.Cnt_D + Sipa
                   End
     If Dat=2 Then Do
                    Sipa_I = Sipa_I + Sipa
                    Tot_AllocI.Cnt_I = Tot_AllocI.Cnt_I + Sipa
                   End
   End
If (Index(MartapV.u,'SPACE-TYPE-') <> Ø) Then
          Do
            Space_Type    = Substr(MartapV.u,18,14)
            Space_Type    = Translate(Space_Type,'','-')
            If Space_Type = CYLINDER Then Space_Type=CYL
            If Space_Type = TRACK    Then Space_Type=TRK
            If Dat = 1 Then Space_TypeD = Space_Type
            If Dat = 2 Then Space_TypeI = Space_Type
          End
If (Index(MartapV.u,'SPACE-PRI—') <> Ø) Then
  Do
   If Dat = 1 Then
       Do
         Space_PrimD = Substr(MartapV.u,17,16)
         Space_PrimD = Translate(Space_PrimD,'','-')
       End
   If Dat = 2 Then
       Do
         Space_PrimI = Substr(MartapV.u,17,16)
         Space_PrimI = Translate(Space_PrimI,'','-')
       End
  End
```

```
If (Index(MartapV.u,'SPACE-SEC—') <> Ø) Then
  Do
   If Dat = 1 Then
       Do
         Space_SecD = Substr(MartapV.u,24,9)
         Space_SecD = Translate(Space_SecD,'','-')
       End
   If Dat = 2 Then
       Do
         Space_SecI = Substr(Martapv.u,24,9)
         Space_SecI = Translate(Space_SecI,'','-')
       End
  End
END
/*            CHECK IF IT'S A MULTI-VOLUME DATASET ?                  */
/* To determine if dataset is multi-volume one, we could check the    */
/* Sysreason value of LISTDSI function. If Sysreason is 19 then it'd  */
/* be a multi-volume dataset.                                         */
/* However,if a multi-volume dset, at the moment, resides on a single */
/* volume, then this function can't figure it out and assumes that    */
/* it's a single volume dataset. For this reason, here we take into   */
/* account the number of "VOLSER—" strings in the LISTCAT output.     */
/* If there are at least two strings then that dataset will regarded  */
/* as a multi-volume dataset.                                         */
/* Note: Data component & Index component volumes are counted         */
/* separately. So, in order a VSAM dset to be a multi-volume, total   */
/* volume count should be greater than 2.                             */
If Vsam=1 Then
  Do
    If Cnt2>2 Then Msg2="Multi-volume  dataset"
              Else Msg2="Single volume dataset"
    Cnt = Cnt_D","Cnt_I
  End
         Else
  Do
    If (Cnt2>1 & Gdg=Ø) Then Msg2="Multi-volume  dataset"
                        Else Msg2="Single volume dataset"
  End
RETURN   /* End-of-the-procedure-VOLCNT */
PROCVSAM:
Msg1="VSAM data set."
Mig=1
Do Nu = Ø to 1Ø*(((Cnt_D+Cnt_I)%1Ø)) by 1Ø
   Pnl = "Pan"Nu
   If Nu = Ø Then Flg = 1;Else Flg = Ø
   If Cnt >= Nu Then
                Do
                 Call Process_Vsam
                 Call Panel_Vsam
                End
```

```
End
RETURN /* End-of-the-procedure-PROCVSAM */
PROCESS_VSAM:
X = Outtrap('PernasV.')
"Listc Entry("Fi") Allocation"
X = Outtrap('Off')
y = Ø /* This variable is used to find total volume count. */
Do c=1 to PernasV.Ø
If Index(Substr(PernasV.c,1,2Ø),'DATA ——') <> Ø Then Dat = 1
If Index(Substr(PernasV.c,1,2Ø),'INDEX ——') <> Ø Then Dat = 2
If (Index(PernasV.c,'VOLSER——') <> Ø) &,
    (Index(PernasV.c,'VOLSER————————*')) =Ø Then
    Do
     y = y + 1                          /* Increment Volume_count */
     VolserDI= Substr(PernasV.c,26,6)
     Ext     = Substr(PernasV.c,116,3)
     Ext     = Translate(Ext,'','-')
     kØ   = Space_TypeD","Space_TypeI
     If Dat = 1 Then
     DO
      Component = Data
      TotA=Sipa_D
      TotE_D = TotE_D + Ext
      Top=VolserDI||" "||Tot_AllocD.y||" "||Component||"
"||Space_PrimD||"
"||,
      Space_SecD||" "||Ext
     END
     If Dat = 2 Then
     DO
      Component = Index
      TotU=Sipa_I
      TotE_I = TotE_I + Ext
      a = y-Cnt_D
      Top=VolserDI||" "||Tot_AllocI.a||" "||Component||"
"||Space_PrimI||"
"||,
      Space_SecI||" "||Ext
     END
     TotE = TotE_D","TotE_I
     If y = 1 Then Do
                   Parse VAR Top k1 k2 k3 k4 k5 k6
                   "Ispexec Vput (kØ k1 k2 k3 k4 k5 k6) Profile"
                   End
     If y = 2 Then Do
                   Parse VAR Top k7 k8 k9 k1Ø k11 k12
                   "Ispexec Vput (kØ k7 k8 k9 k1Ø k11 k12) Profile"
                   End
     If y = 3 Then Do
                   Parse VAR Top k13 k14 k15 k16 k17 k18
```

```
                  "Ispexec Vput (k13 k14 k15 k16 k17 k18) Profile"
                  End
     If y = 4 Then Do
                  Parse VAR Top k19 k2Ø k21 k22 k23 k24
                  "Ispexec Vput (k19 k2Ø k21 k22 k23 k24) Profile"
                  End
     If y = 5 Then Do
                  Parse VAR Top k25 k26 k27 k28 k29 k3Ø
                  "Ispexec Vput (k25 k26 k27 k28 k29 k3Ø) Profile"
                  End
     If y = 6 Then Do
                  Parse VAR Top k31 k32 k33 k34 k35 k36
                  "Ispexec Vput (k31 k32 k33 k34 k35 k36) Profile"
                  End
     If y = 7 Then Do
                  Parse VAR Top k37 k38 k39 k4Ø k41 k42
                  "Ispexec Vput (k37 k38 k39 k4Ø k41 k42) Profile"
                  End
     If y = 8 Then Do
                  Parse VAR Top k43 k44 k45 k46 k47 k48
                  "Ispexec Vput (k43 k44 k45 k46 k47 k48) Profile"
                  End
     If y = 9 Then Do
                  Parse VAR Top k49 k5Ø k51 k52 k53 k54
                  "Ispexec Vput (k49 k5Ø k51 k52 k53 k54) Profile"
                  End
     If y =1Ø Then Do
                  Parse VAR Top k55 k56 k57 k58 k59 k6Ø
                  "Ispexec Vput (k55 k56 k57 k58 k59 k6Ø) Profile"
                  End
    End
END
"Ispexec Vput (TotA TotU TotE Fi y) Profile"
RETURN   /* End-of-the-procedure-PROCESS_VSAM */
PANEL_VSAM:
Address Tso
Ddname='Martapv'
If Flg=1 Then "Alloc Fi("Ddname") Reuse New Del Dso(Po) Dir(1) Sp(1)" ,
              "Track Recfm(F B) Lrecl(8Ø) Unit(Sysda)"
         Else "Alloc Fi("Ddname") Reuse Shr"
Address Ispexec
"Lminit Dataid(Did) Ddname("Ddname") Enq(Exclu)"
"Lmopen Dataid(&Did) Option(Output)"
Call Put ")ATTR                                              "
Call Put " _ TYPE(TEXT)   COLOR(WHITE) CAPS(OFF) HILITE(USCORE)"
Call Put " @ TYPE(TEXT)   COLOR(RED)   CAPS(OFF) HILITE(USCORE)"
Call Put " ! TYPE(TEXT)   COLOR(YELLOW) JUST(RIGHT)          "
Call Put " + TYPE(TEXT)   COLOR(TURQ)  CAPS(OFF) JUST(LEFT)    "
Call Put " ? TYPE(TEXT)   COLOR(BLUE)  CAPS(OFF) HILITE(REVERSE)"
Call Put " / TYPE(TEXT)   COLOR(RED)   HILITE(REVERSE)"
```

```
Call Put " $ TYPE(OUTPUT) COLOR(GREEN) CAPS(OFF)"
Call Put " [ TYPE(OUTPUT) COLOR(PINK)  CAPS(OFF) JUST(RIGHT)   "
Call Put " ] TYPE(OUTPUT) COLOR(BLUE)  CAPS(OFF) JUST(RIGHT)   "
Select
When (Cnt = 1Ø)         Then Call Put ')BODY WINDOW(57,22)'
When ((Cnt>1Ø) & y > 9) Then Call Put ')BODY WINDOW(57,18)'
When ((Cnt>1Ø) & y <=9) Then Call Put ')BODY WINDOW(57,'y+11')'
When (Cnt<1Ø)           Then Call Put ')BODY WINDOW(57,'y+11')'
End
Call Put "?           MULTI VOLUME DATASET DETAILS     "
Call Put "+                                             "
Call Put "!Data set:$Z"Substr(Blank,45-Length(Fi))"+"
Call Put "/" Msg2                          "/" Msg1
Call Put "+Volume  Alloc   Comp  Primary   Second    Numext  Vseq"
If y>Ø Then Call Put "[Z     [Z     [Z     [Z       [Z       [Z       !"
Nu+1
If y>1 Then Call Put "[Z     [Z     [Z     [Z       [Z       [Z       !"
Nu+2
If y>2 Then Call Put "[Z     [Z     [Z     [Z       [Z       [Z       !"
Nu+3
If y>3 Then Call Put "[Z     [Z     [Z     [Z       [Z       [Z       !"
Nu+4
If y>4 Then Call Put "[Z     [Z     [Z     [Z       [Z       [Z       !"
Nu+5
If y>5 Then Call Put "[Z     [Z     [Z     [Z       [Z       [Z       !"
Nu+6
If y>6 Then Call Put "[Z     [Z     [Z     [Z       [Z       [Z       !"
Nu+7
If y>7 Then Call Put "[Z     [Z     [Z     [Z       [Z       [Z       !"
Nu+8
If y>8 Then Call Put "[Z     [Z     [Z     [Z       [Z       [Z       !"
Nu+9
If y>9 Then Call Put "[Z     [Z     [Z     [Z       [Z       [Z       !"
Nu+1Ø
Call     Put "+"
If y <= 9 | (Cnt=1Ø) Then
  Do
    Call Put "+Space unit (D,I):]Z      +  SPACE USAGE    : (in
tracks)"
    Call Put "+Tot.extents(D,I):]Z      +  Total alloc (D):]Z         +"
    Call Put "+Num.of vol.(D,I):]Z      +  Total alloc (I):]Z         +"
    Call Put "+"
    Call Put "_Hit@<Enter>_to exit.+"
  End
  Else Call Put "_Hit@<Enter>_to view the next page.+"
Call Put ")Init                                          "
Call Put ".Zvars='(Fi                                   +"
If y> Ø Then Call Put "         k1 k2 k3 k4 k5 k6       +"
If y> 1 Then Call Put "         k7 k8 k9 k1Ø k11 k12    +"
If y> 2 Then Call Put "         k13 k14 k15 k16 k17 k18 +"
```

```
If y> 3 Then Call Put "         k19 k2Ø k21 k22 k23 k24 +"
If y> 4 Then Call Put "         k25 k26 k27 k28 k29 k3Ø +"
If y> 5 Then Call Put "         k31 k32 k33 k34 k35 k36 +"
If y> 6 Then Call Put "         k37 k38 k39 k4Ø k41 k42 +"
If y> 7 Then Call Put "         k43 k44 k45 k46 k47 k48 +"
If y> 8 Then Call Put "         k49 k5Ø k51 k52 k53 k54 +"
If y> 9 Then Call Put "         k55 k56 k57 k58 k59 k6Ø +"
If y <= 9 | (Cnt=1Ø)  Then Call Put " kØ TotE TotA Cnt TotU)'"
                      Else Call Put "                      )'"
Call Put ")Proc                  "
Call Put ")End                   "
"Lmmadd Dataid(&Did) Member("Pnl")"
"Lmfree Dataid(&Did)"
"Libdef Ispplib Library Id("Ddname")"
"Addpop"
"Display Panel("Pnl")"
"Rempop"
"Libdef Ispplib"
RETURN /* End-of-the-procedure-PANEL_VSAM*/
PROCNVSAM:
Str1= FI "Smsinfo Norecall" /* In case a migrated dset, don't recall. */
x = Listdsi(Str1)
If SYSREASON=3Ø Then Sms=Ø                /* If Sms=Ø then No-SMS    */
                Else Sms=1                /* If Sms=1 then SMS dset. */
Msg1 = Sysdsorg " data set."
If (Pdse=1) ³ (Hfs=1) ³ (Gdg=1) Then Mig=1
If Gdg=1  Then Do;Msg1="GDG  data set.";Cnt=Ø;End
If Pdse=1 Then    Msg1="PDSE data set."
If Hfs=1  Then    Msg1="HFS  data set."
If SYSREASON =  5 Then Do;If Gdg=Ø Then
                          Msg1="Uncatalogued data set";Mig=1;End
If SYSREASON =  8 Then Do;Msg1="Tape dataset    "   ;Mig=1;Cnt=Ø;End
If SYSREASON =  9 Then Do;Msg1="Migrated dataset"   ;Mig=1;End
/* Panel handling. At most, 5 panels can be automatically built and   */
/* then displayed. (Each panel can have only 1Ø volume details and    */
/* maximum number of volumes that can be used to store a multi-volume */
/* data set is 59. For this reason, 5 panels will be enough.          */
Do Nu = Ø to 1Ø*((Cnt%1Ø)) by 1Ø
      Pnl = "Pan"Nu
      /* Flg variable is used to create the temporary Panel    */
      /* library only once. If Flg=Ø then it'll be created and */
      /* allocated. If Flg=1 then it'll be only allocated.     */
      If Nu = Ø Then Flg = 1;Else Flg = Ø
      If Cnt >= Nu Then
                  Do
                    Call Process_NonVsam
                    Call Panel_Nvsam
                  End
End
RETURN /* End-of-the-procedure-PROCNVSAM */
```

```
PROCESS_NONVSAM:
If Sms = 1 Then ui = 11            /* Listc output for SMS-managed    */
           Else ui = 7             /* dataset differs from output     */
                                   /* for non-SMS managed datasets.   */
y = Ø /* This variable is used to find total volume count.            */
Do u = Nu+ui to Martapv.Ø     /* Do-2    */
Volser=Substr(Martapv.u,26,6)
ju=Index(Martapv.u,'VOLSER—')
If (ju <> Ø & Volser <> '——*') Then
Do                            /* Do-1    */
  If Mig<>Ø Then y = 1
            Else y = y + 1        /* Increment Volume_count   */
  Str2 = FI "Volume("Volser")"
  x    = Listdsi(Str2)
  kØ   = Sysunits /* Space units: CYLINDER, TRACK, BLOCK */
  If Pdse=1 Then Sysused = Sysusedpages||"p."
  Top=Volser||" "||SYSALLOC||" "||SYSUSED||" "||SYSPRIMARY||" "||,
    SYSSECONDS||" "||SYSEXTENTS
  If y = 1 Then Do
                 Parse VAR Top k1 k2 k3 k4 k5 k6
                 If Mig<>Ø Then Do
                            If Gdg=1 Then k1="N/A"
                            TotA="N/A";TotU="N/A";TotE="N/A";kØ="N/A"
                            End
                   Else Do
                          TotA = TotA + k2;TotU = TotU + k3;TotE=TotE+k6
                          "Ispexec Vput (kØ k1 k2 k3 k4 k5 k6) Profile"
                         End
               End
  If y = 2 Then Do;Parse VAR Top k7  k8  k9  k1Ø k11 k12
                   TotA = TotA + k8;TotU = TotU + k9;TotE=TotE+k12
                   "Ispexec Vput (k7 k8 k9 k1Ø k11 k12) Profile";End
  If y = 3 Then Do;Parse VAR Top k13 k14 k15 k16 k17 k18
                   TotA = TotA + k14;TotU = TotU + k15;TotE=TotE+k18
                   "Ispexec Vput (k13 k14 k15 k16 k17 k18) Profile";End
  If y = 4 Then Do;Parse VAR Top k19 k2Ø k21 k22 k23 k24
                   TotA = TotA + k2Ø;TotU = TotU + k21;TotE=TotE+k24
                   "Ispexec Vput (k19 k2Ø k21 k22 k23 k24) Profile";End
  If y = 5 Then Do;Parse VAR Top k25 k26 k27 k28 k29 k3Ø
                   TotA = TotA + k26;TotU = TotU + k27;TotE=TotE+k3Ø
                   "Ispexec Vput (k25 k26 k27 k28 k29 k3Ø) Profile";End
  If y = 6 Then Do;Parse VAR Top k31 k32 k33 k34 k35 k36
                   TotA = TotA + k32;TotU = TotU + k33;TotE=TotE+k36
                   "Ispexec Vput (k31 k32 k33 k34 k35 k36) Profile";End
  If y = 7 Then Do;Parse VAR Top k37 k38 k39 k4Ø k41 k42
                   TotA = TotA + k38;TotU = TotU + k39;TotE=TotE+k42
                   "Ispexec Vput (k37 k38 k39 k4Ø k41 k42) Profile";End
  If y = 8 Then Do;Parse VAR Top k43 k44 k45 k46 k47 k48
                   TotA = TotA + k44;TotU = TotU + k45;TotE=TotE+k48
                   "Ispexec Vput (k43 k44 k45 k46 k47 k48) Profile";End
```

```
  If y = 9 Then Do;Parse VAR Top k49 k5Ø k51 k52 k53 k54
                   TotA = TotA + k5Ø;TotU = TotU + k51;TotE=TotE+k54
                   "Ispexec Vput (k49 k5Ø k51 k52 k53 k54) Profile";End
  If y = 1Ø Then Do;Parse VAR Top k55 k56 k57 k58 k59 k6Ø
                   TotA = TotA + k56;TotU = TotU + k57;TotE=TotE+k6Ø
                   "Ispexec Vput (k55 k56 k57 k58 k59 k6Ø) Profile";End
End /* End Do-1    */
End  /* End Do-2    */
"Ispexec Vput (TotA TotU TotE Fi y) Profile"
RETURN   /* End-of-the-procedure-PROCESS_NONVSAM */
PANEL_NVSAM:
Address Tso
Ddname='Martapv'
If Flg=1 Then "Alloc Fi("Ddname") Reuse New Del Dso(Po) Dir(1) Sp(1)" ,
              "Track Recfm(F B) Lrecl(8Ø) Unit(Sysda)"
         Else "Alloc Fi("Ddname") Reuse Shr"
Address Ispexec
"Lminit Dataid(Did) Ddname("Ddname") Enq(Exclu)"
"Lmopen Dataid(&Did) Option(Output)"
Call Put ")ATTR                                                  "
Call Put " _ TYPE(TEXT)   COLOR(WHITE) CAPS(OFF) HILITE(USCORE)"
Call Put " @ TYPE(TEXT)   COLOR(RED)   CAPS(OFF) HILITE(USCORE)"
Call Put " ! TYPE(TEXT)   COLOR(YELLOW) JUST(RIGHT)              "
Call Put " + TYPE(TEXT)   COLOR(TURQ)  CAPS(OFF) JUST(LEFT)    "
Call Put " ? TYPE(TEXT)   COLOR(BLUE)  CAPS(OFF) HILITE(REVERSE)"
Call Put " / TYPE(TEXT)   COLOR(RED)   HILITE(REVERSE)"
Call Put " $ TYPE(OUTPUT) COLOR(GREEN) CAPS(OFF)"
Call Put " [ TYPE(OUTPUT) COLOR(PINK)  CAPS(OFF) JUST(RIGHT)   "
Call Put " ] TYPE(OUTPUT) COLOR(BLUE)  CAPS(OFF) JUST(RIGHT)   "
Select
When (Cnt = 1Ø)         Then Call Put ')BODY WINDOW(57,22)'
When ((Cnt>1Ø) & y > 9) Then Call Put ')BODY WINDOW(57,18)'
When ((Cnt>1Ø) & y <=9) Then Call Put ')BODY WINDOW(57,'y+11')'
When (Cnt<1Ø)           Then Call Put ')BODY WINDOW(57,'y+11')'
End
Call Put "?           MULTI VOLUME DATASET DETAILS     "
Call Put "+                                             "
Call Put "!Data set:$Z"Substr(Blank,45-Length(Fi))"+"
Call Put "/" Msg2                         "/" Msg1
If y <> Ø Then
      Call Put "+Volume  Alloc   Used   Primary   Second   Numext
Vseq"
      Else Call Put "+"
If y>Ø Then Call Put "[Z     [Z     [Z      [Z       [Z       [Z      !"
Nu+1
If y>1 Then Call Put "[Z     [Z     [Z      [Z       [Z       [Z      !"
Nu+2
If y>2 Then Call Put "[Z     [Z     [Z      [Z       [Z       [Z      !"
Nu+3
If y>3 Then Call Put "[Z     [Z     [Z      [Z       [Z       [Z      !"
```

```
Nu+4
If y>4 Then Call Put "[Z     [Z     [Z      [Z       [Z       [Z      !"
Nu+5
If y>5 Then Call Put "[Z     [Z     [Z      [Z       [Z       [Z      !"
Nu+6
If y>6 Then Call Put "[Z     [Z     [Z      [Z       [Z       [Z      !"
Nu+7
If y>7 Then Call Put "[Z     [Z     [Z      [Z       [Z       [Z      !"
Nu+8
If y>8 Then Call Put "[Z     [Z     [Z      [Z       [Z       [Z      !"
Nu+9
If y>9 Then Call Put "[Z     [Z     [Z      [Z       [Z       [Z      !"
Nu+1Ø
Call Put "+"
If y <= 9 | (Cnt=1Ø) Then
  Do
    Call Put "+Space unit     : ]Z       + Total extents   :]Z    +"
    Call Put "+Space allocated: ]Z       + Num. of volumes :]Z    +"
    Call Put "+Space used     : ]Z       +                         +"
    Call Put "+"
    Call Put "_Hit@<Enter>_to exit.+"
  End
  Else Call Put "_Hit@<Enter>_to view the next page.+"
Call Put ")Init                                          "
Call Put ".Zvars='(Fi                                   +"
If y> Ø Then Call Put "         k1 k2 k3 k4 k5 k6       +"
If y> 1 Then Call Put "         k7 k8 k9 k1Ø k11 k12    +"
If y> 2 Then Call Put "         k13 k14 k15 k16 k17 k18 +"
If y> 3 Then Call Put "         k19 k2Ø k21 k22 k23 k24 +"
If y> 4 Then Call Put "         k25 k26 k27 k28 k29 k3Ø +"
If y> 5 Then Call Put "         k31 k32 k33 k34 k35 k36 +"
If y> 6 Then Call Put "         k37 k38 k39 k4Ø k41 k42 +"
If y> 7 Then Call Put "         k43 k44 k45 k46 k47 k48 +"
If y> 8 Then Call Put "         k49 k5Ø k51 k52 k53 k54 +"
If y> 9 Then Call Put "         k55 k56 k57 k58 k59 k6Ø +"
If y <= 9 | (Cnt=1Ø)  Then Call Put " kØ TotE TotA Cnt TotU)'"
                      Else Call Put "                     )'"
Call Put ")Proc                  "
Call Put ")End                   "
"Lmmadd Dataid(&Did) Member("Pnl")"
"Lmfree Dataid(&Did)"
"Libdef Ispplib Library Id("Ddname")"
"Addpop"
"Display Panel("Pnl")"
"Rempop"
"Libdef Ispplib"
RETURN /* End-of-the-procedure-PANEL_NVSAM */
PUT:
Parse Arg Prm    /* Prm : Record to be written */
```

```
"Lmput Dataid(&Did) Mode(Invar) Dataloc(Prm) Datalen(8Ø)"
RETURN /* End-of-the-procedure-PUT    */
```

Atalay Gul
Systems Programmer
EDS Barcelona (Spain)

Maintaining a DASD configuration – revisited

In the article entitled *Maintaining a DASD configuration*, in *MVS Update*, Issue 187, April 2002, I unfortunately neglected to include the actual bit of REXX that generates the back-ups. It's called GENBKUPS.

GENBKUPS

```
/* ============================================================ */
/*  GENBKUPS: Create Full Pack DFDSS back-ups suites, using the */
/*           RECDSKM member as source.                          */
/*           Also creates the 'DEF GDG' statements for each     */
/*           of the back-up suites in 'BQIBIØ6.OPSREC.GDGS', in */
/*           the same member name as is used for the back-ups   */
/*           (ie BCKMVSx, where 'x' is the suite suffix).       */
/*           Note that 'BCKMVS@' is a special suite which will  */
/*           be generated containing any disks which have a '?' */
/*           in 'RECDSKM'. This implies that the disk has been  */
/*           initialized, but that no BCKMVS- suite has yet     */
/*           been assigned to it.                               */
/*                                                              */
/*           If the 'gen_jobs' variable is set to 'Y' then the  */
/*           actual back-up JCL will be created, allowing a job */
/*           scheduler to submit the back-ups, instead of the   */
/*           Operators via panels. Note that this function is   */
/*           only valid when a parm of 'B' is passed, implying  */
/*           we are running in batch, as the process is quite   */
/*           slow.                                              */
/* ============================================================ */
/* Trace ir */
Parse Upper Arg parm .
/* These are valid names which may be used for back-up suites.  */
bkups = "A B C D E F G H I J K L M N O P Q R S T U V W X Y Z Ø 1 2"
bkups = bkups" 3 4 5 6 7 8 9 @"
gen_jobs = "N"                  /* Do/Don't generate actual jobs  */
entries = Words(bkups)              /* Number of entries to init  */
badones = "N"                       /* No dodgy ones found (yet)  */
```

```
df1 = " DEF GDG(NAME(SYS8.&SYS..BCKMVS"
df2 = ") LIMIT(2) SCRATCH NOEMPTY)"
code = Ø
If parm = "B" Then Do
  Say ">>> Starting Backup Suite, GDG Base and Fullpack JCL Generation:"
  Say ">>> ============================================================="
End
Else Do
  Say ">>> Starting Backup Suite and GDG Base Generation:"
  Say ">>> ==========================================="
End
Do a = 1 to entries
  suffx = Word(bkups,a)             /* Get each suffix            */
  Interpret "MVS"suffx".Ø = ''"     /* Initialize all tables...   */
  Interpret "MVS"suffx"BKP = Ø"     /* ...total disks for suite   */
End
/* Read the "BQIBIØ6.OPSREC.CONTROL(RECDSKM)" member, and       */
/* create the relevant data in "BQIBIØ6.OPSREC.BACKUPS".        */
Address "TSO"
"ALLOC FI(INPUT) DA('BQIBIØ6.OPSREC.CONTROL(RECDSKM)') SHR"
"EXECIO * DISKR INPUT (Stem inpt. FINIS"
"FREE FI(INPUT)"
/* Read in the data from "RECDSKM" and split out into relevant   */
/* back-up suite stem variables (format is "MVSx.", where 'x' is */
/* the suite name from the table above)...                       */
Do a = 1 to inpt.Ø
  Parse Upper Var inpt.a addr volid type suite com recstr .
  If suite = "NONE" Then      /* No back-ups if not required        */
    Iterate
  If addr = "*" Then          /* Ignore comments cards              */
    Iterate
  If addr = "ADDR" Then       /* Ignore headers                     */
    Iterate
  suffx = Right(suite,1)
  If suffx = "?" Then Do
    suffx = "@"
    badones = "Y"
  End
  Interpret "MVS"suffx"BKP = MVS"suffx"BKP + 1"
  devt = "xxxx"
  If type = "93" Then devt = "339Ø"
  If type = "8Ø" Then devt = "338Ø"
  If type = "81" Then devt = "338Ø"
  If type = "82" Then devt = "338Ø"
  Interpret "VOL"suffx".MVS"suffx"BKP = volid addr devt com"
  defgdg = df1¦¦suffx"."volid¦¦df2
  Interpret "DEF"suffx".MVS"suffx"BKP = defgdg"
End
/* If any entries have been created for a back-up suite then write  */
/* them to the relevant member...                                   */
```

```
Do a = 1 to entries
  suffx = Word(bkups,a)             /* Get each suffix            */
  Interpret "bkct = MVS"suffx"BKP"  /* Get count for this suffix  */
  If bkct > 6Ø Then                 /* Over 6Ø in suite is no good*/
    Call TOO_MANY
  Interpret "MVS"suffx".Ø = MVS"suffx"BKP"
  If bkct <> Ø Then Do              /* If any records for suffix: */
    suite = "BCKMVS"suffx
    comnt = ""
    If parm = "B" Then
      comnt = "(FULLPACK JCL)"
    Say Left(">>> Generating "suite", volumes = "bkct,45)||comnt
    "ALLOC FI("suite") DA('BQIBIØ6.OPSREC.BACKUPS("suite")') SHR"
    "ALLOC FI("gdgdf") DA('BQIBIØ6.OPSREC.GDGS("suite")') SHR"
    stemid = "VOL"suffx"."
    "EXECIO * DISKW "suite " (Stem "stemid" FINIS"
    stemid = "DEF"suffx"."
    "EXECIO * DISKW "gdgdf " (Stem "stemid" FINIS"
    "FREE FI("suite")"
    "FREE FI("gdgdf")"
    If parm = "B" Then              /* Only in batch, and...      */
    If gen_jobs = "Y"               /* only if flag set           */
    Then Do
      x = OUTTRAP("nulls.",2Ø,"NOCONCAT")   /* Suppress next msg  */
      "DELETE 'BQIBIØ6.OPSREC.FULLPACK.BCKMVS"suffx"'"
      x = OUTTRAP("OFF")
      "ALLOC F(BKJCL) DA('BQIBIØ6.OPSREC.FULLPACK.BCKMVS"suffx"')
        BLKSIZE(616Ø) DSORG(PS) UNIT(SYSDA) CYL RECFM(F B) SPACE(2 1)
        LRECL(8Ø) DIR(43) NEW CATALOG"
      "FREE F(BKJCL)"
      Do nn = 1 to bkct             /* Create JCL for each volume */
        stemid = "VOL"suffx"."nn
        Interpret "Parse Upper Var "stemid" disvol disadr disdev ."
        bkjbnm = "£GOI"suffx"B"Right("Ø"nn,2)  /* eg £GOIABØ1!  */
        suite = "BCKMVS"suffx
        usrid = "&USRID"
        "ALLOC F(ISPFILE) DA('BQIBIØ6.OPSREC.FULLPACK.BCKMVS"suffx"')
          SHR"
        ADDRESS "ISPEXEC"
        "FTINCL SOPBK17Ø"           /* !!! NOTE this skeleton is  */
                                    /* !!! shared with the ISPF   */
                                    /* !!! based back-ups system. */
        If rc <> Ø Then
          Say ">> Error including skeleton SOPBK17Ø..."
        "FTCLOSE NAME("disvol")"
        ADDRESS "TSO"
        "FREE F(ISPFILE)"
      End                                   /* Do nn = 1 to... */
    End                                     /* If gen_jobs = Y */
  End                                       /* If bkct <> Ø... */
```

```
End                                          /* Do a = 1 to...  */
/* A list of ALL of the back-ups is written to member 'COMDBKPS'.   */
/* In this list we insert the suite name and the decimal value of   */
/* the hex address (this allows us to easily sort into hex order).  */
/* This list is a complete list of ALL volumes that are sent to     */
/* BKUPSITE, irrespective of whether they are actually restored or  */
/* not (those with an 'X' in the BKUPSITE Flag field are not norm-  */
/* ally restored). The "FAILMAST" Exec will later be used to create */
/* a listing of disks required, by MVS system (which excludes those */
/* not normally restored), and a list of ALL disks that we send     */
/* backups for. These lists are e-mailed to BKUPSITE so that they   */
/* can create the necessary profiles for tests and also have a full */
/* list in case we need to invoke recovery.                         */
Do a = 1 to entries
  suffx = Word(bkups,a)             /* Get each suffix            */
  Interpret "xx = MVS"suffx"BKP"    /* Get count for this suffix  */
  Interpret "MVS"suffx".Ø = MVS"suffx"BKP"
  If xx <> Ø Then Do w = 1 to xx    /* If any records for suffix: */
    Interpret "TEMP = VOL"suffx"."w
    suite = "BCKMVS"suffx
    Parse Var temp vol hexaddr devt comflag .
    If comflag <> "N" Then Do
      sortfld = Right('  'X2D(hexaddr),6)
      Queue " "vol" "hexaddr" "devt" "suite" "comflag"       "sortfld
    End
  End
End
num = Queued()
"ALLOC FI(COMDBKP) DA('BQIBIØ6.OPSREC.BACKUPS(COMDBKPS)') SHR"
"EXECIO "num" DISKW COMDBKP (FINIS"
"FREE FI(COMDBKP)"
If badones = "Y" Then Do
  Say "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"
  Say "++ Invalid backup suite(s) found - please resolve and rerun ++"
  Say "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"
  code = 8
End
Exit(code)
TOO_MANY:
Say " "
Say ">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<"
Say ">>>       Over 6Ø back-ups found in suite BCKMVS"suffx" <<<"
Say ">>>         Back-up suite creation terminated...        <<<"
Say ">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<"
Say " "
Exit 16
```

Grant Carson
Systems Programmer (UK)

Overview of ISPF panel processing commands

During the evolution of ISPF Dialog Manager, many useful features have been added, in particular in the area of panel processing. My experience has shown that many of these features are either unknown or not fully appreciated. The purpose of this article is to provide an overview of some of the most useful of these features as they affect panel processing. The correct operation of some features depends on other settings being made. This article discusses such dependencies.

In its most simple form, a panel displays information, allows information to be input, and can perform simple formal validations on the entered data – for example, is a dataset name formally correct (not longer than 44 characters, the parts of a dataset name are not longer than eight characters, alphanumeric with a leading alphabetic character, and separated with a period)?

Originally, any more intensive processing had to be performed in the invoking procedure (REXX or CLIST) or program. Features added to Dialog Manager now provide more flexibility regarding how external procedures can be activated. Some facilities allow procedures to be invoked in parallel with the display and return control to the panel display. Such facilities can be used, for example, to verify the physical presence of a dataset, and, if it is not present, allow the panel to display an error message immediately.

This article describes the various means of invoking procedures (commands) from a panel. It also provides an example that uses all these facilities.

The methods of invoking a procedure from a panel are:

- Set a command variable
- Invoke a panel exit
- Action bar
- Command invoked by a PF key.

SET A COMMAND VARIABLE

If the displayed panel explicitly sets a command variable (normally ZCMD, or the field name specified in the CMD option of the *)BODY* header), the procedure or program that displayed the panel can execute the command at the end of the display.

For example:

```
ADDRESS TSO
ADDRESS ISPEXEC "DISPLAY PANEL(PN)"
IF RC = Ø & zcmd <> '' THEN INTERPRET zcmd
```

This method has always been available.

INVOKE A PANEL EXIT

A panel exit can invoke either a program or a REXX EXEC. Although a panel exit cannot itself invoke any ISPF services, it can use operating system services, TSO services, etc. A panel exit also has the restriction that, although it can be passed and return ISPF variables, it cannot alter their length. This means that any variables passed to a panel exit must be passed with their maximum possible length, ie padded where necessary. A panel exit returns immediately after the point of execution.

For example, a panel exit could be passed two variables – a dataset name and a status field. The status field could be set to an appropriate code if the dataset does not exist.

Since the availability of panel exits, it has always been possible to invoke a program, either by name (PGM keyword) or by its address (LOAD keyword). The ability to invoke a REXX EXEC has only recently become available, although I published a program in *MVS Update* in January 1993 that could be used to invoke a REXX EXEC.

For example:

```
&MSG = '                          ' /* clear 32 bytes */
PANEXIT ((FILENAME,MSG),REXX,XPXCMD2)
IF (&MSG NE ' ')
  &LMSG = &MSG
  .MSG = XXMSGØØ5
  EXIT
```

In this case, two variables (FILENAME,MSG) are passed to the REXX

EXEC XPXCMD2. Because a panel exit cannot alter the length of passed variables, MSG must be initialized to the maximum possible length. The called REXX EXEC receives a single parameter that contains the address and length of the passed variables.

IBM supplies the ISPREXPX function that a panel exit can use to set local REXX variables ('I'-option) or ISPF variables ('T'); the REXX STORAGE instruction can also be used to access and return the passed ISPF variables.

For example:

```
/* REXX panel exit */
CALL ISPREXPX('I') /* set local REXX variables */
/* varnames.1: FILENAME */
/* varlens.1: length of FILENAME */
/* varnames.2: MSG */
/* varlens.2: length of MSG */
txt = SYSDSN(filename)
IF txt <> 'OK',
  THEN msg = LEFT(txt,varlens.2) /* set message text */
CALL ISPREXPX('T') /* set ISPF variables */
```

Although the diagram shows a panel exit being invoked from the *)PROC* section, panel exits can also be invoked from the *)INIT* and *)REINIT* sections.

ACTION BAR

When an action bar is opened, it lists the associated actions. The associated action for a selected action bar entry can specify either the invocation of a command or the setting of a variable. To illustrate the techniques involved, the example uses both types of action. When a command is invoked, the associated Command Table entry specifies the processing to be performed. If a CMD (command) is assigned, the associated command will run in a separate window. After being processed, the command returns to the start of the *)PROC* section. If no Command Table entry exists, ZCMD will be set to the specified command; this command is then processed as described in the first method.

For example:

```
)ABC DESC(File)
 PDC DESC('New')
   ACTION RUN(XACMD) PARM(&FILENAME)
 PDC DESC('Old')
)ABCINIT
 .ZVARS = PDC
 &PDC = ' '
)ABCPROC
 IF (&PDC EQ 2)
   &ABCMD = 'memname = XBCMD(&filename)'
```

The *)ABC* section specifies the action bar entry (here File). The subsequent PDC entries specify the pull-down choices, here New and Old. The ACTION entry for New specifies the command to be executed when it is selected. The *)ABCPROC* section specifies the processing to be performed when the variable associated with the pull-down (PDC) is set to 2, ie Old selected.

COMMAND INVOKED BY A PF KEY

The effect of a command invoked by a PF key does not actually differ from a command explicitly set in the command variable; the difference lies in how the command is set. ISPF, or more correctly PDF, uses PF keys as shortcuts for frequently-used functions, eg PF1 = Help, PF3 = End. An application can define its own application-specific function keys (key-list table) by including a table with the name *appl*KEYS, where *appl* is the name of the application specified by the NEWAPPL parameter when the application is started. The function key table specifies the command associated with the function key, for example KEY4DEF = PROMPT assigns the PROMPT command to the PF4 key. The application command table *appl*CMDS specifies the function associated with the command, for example ZCTVERB=PROMPT, ZCTACT=PASSTHRU means that the PROMPT command is assigned unchanged to the command field.

The command name variable with the associated command in the application command table is passed to the *)PROC* section.

Note: to illustrate the widest range of possibilities, the example specifies PASSTHRU as an action associated with the PROMPT command. Obviously another possibility would be to directly specify ZCTACT=SELECT PCMD(...) as ZCTACT for ZCTVERB=PROMPT,

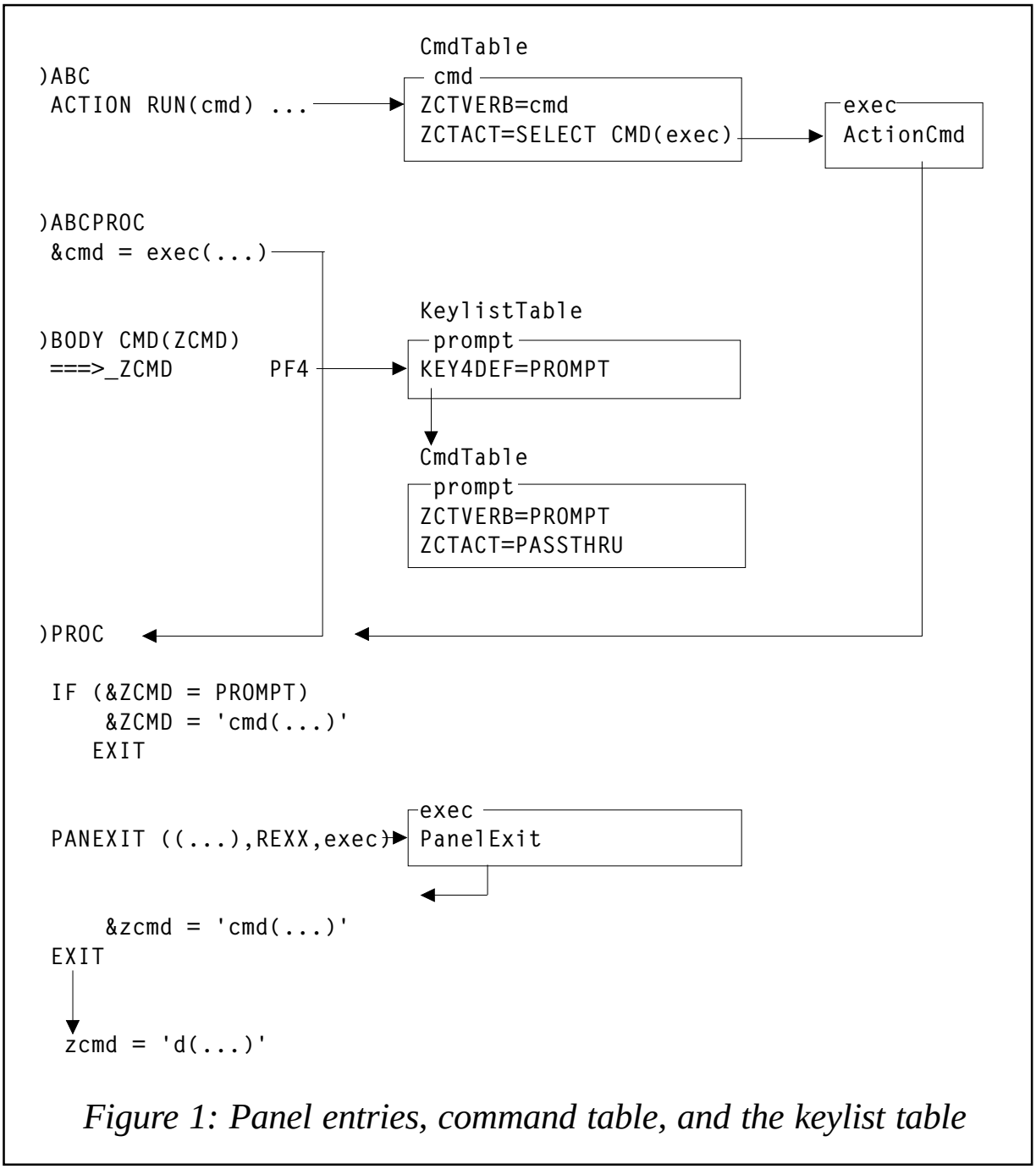

Figure 1: Panel entries, command table, and the keylist table

even though this is not identical because PCMD in this case executes in a separate function pool.

Figure 1 shows the invocation of commands. The Figure illustrates the interaction of panel entries with the command table and the keylist table.

EXAMPLE

The following example is a simple implementation of an end user-oriented application for editing a dataset (EUEDIT); it has been chosen to illustrate the methods described above.

EUEDIT allows the editing of a member of a partitioned dataset. To isolate the user from needing to know how to allocate a dataset, the allocate function is included in the application. Similarly, to reduce the learning curve, the handling is similar to a typical PC application's: the pop-up panel associated with the NEW action allocates a new dataset, the pop-up panel associated with the OPEN action allows the selection of an existing member. The member field on the panel can either be used to enter a new member name or as a prompt field when the PF4 key is pressed with the cursor positioned on the member input field.

To reduce the size of the application, the standard IBM panel (ISRUAASE) is used to specify the dataset details (block size, record length, space, etc). A true end-user-oriented editor would provide a panel more appropriate for an end user, and so reduce the error possibilities. However, even with the standard panel, the application has full control over the input (because this does not directly concern the topics discussed in this article, the example does not place any restrictions on the input).

The standard editor is invoked once the dataset and member name have been entered or selected.

ADEMO (INITIAL PROCEDURE)

```
/* REXX - ADEMO */
ADDRESS ISPEXEC
"SELECT CMD(ADEMO1) NEWAPPL(DEMO)"
```

ADEMO invokes the ADEMO1 command with DEMO as the application identifier; the ADEMO procedure is needed only to set the application identifier. This application identifier is used to access the associated command table (DEMOCMDS).

ADEMO1 (PROCESSING PROCEDURE)

```
/* REXX - ADEMO1 */
/* Processing procedure) */
```

```
ADDRESS ISPEXEC
DO FOREVER  "DISPLAY PANEL(DEMOPAN)"  IF RC > Ø THEN LEAVE
  IF abcmd <> '' THEN zcmd = abcmd
  IF zcmd <> '' THEN INTERPRET zcmd
  /* check for completeness */
  IF (filename <> '') & (memname <> '') THEN DO
    dsn = SPACE(filename"("memname")",Ø)
    "EDIT DATASET("dsn")"
  END
END
```

ADEMO1 performs the actual processing. The panel (DEMOPAN) is displayed until the END key is pressed (pressing the END key sets the RC to 8). The panel processing could set either of the two command variables – ABCMD is the command variable set in the action bar processing, ZCMD is the command variable set when the command is entered directly (or as command assigned to a PF key). The associated command will be executed (with the REXX INTERPRET) if either of these command variables has been set.

The standard ISPF editor (EDIT) is called when the dataset name (FILENAME) and member name (MENNAME) have both been specified.

DEMOPAN (DISPLAY PANEL)

```
)PANEL KEYLIST(DISPLAY,DEMO)
/* DEMOPAN
)ATTR DEFAULT(%~_)
? TYPE(PT)
@ TYPE(AB)
% TYPE(ABSL)
< TYPE(FP)
] TYPE(ET)
[ TYPE(NEF) CAPS(ON)
)ABC DESC(File)MNEM(1)
 PDC DESC('New...  ') MNEM(1)
   ACTION RUN(XACMD) PARM(&FILENAME)
 PDC DESC('&TEXT2') MNEM(1)
)ABCINIT
 .ZVARS = PDC
 &PDC = ' '
)ABCPROC
 IF (&PDC EQ 2)
   &ABCMD = 'memname = XBCMD(&filename)'
)ABC DESC(Help) MNEM(1)
 PDC DESC(TOC)
 PDC DESC(Index)
```

```
)ABCINIT
 .ZVARS = PDC
)BODY EXPAND(//) CMD(ZCMD)
?/-/ Edit Frontend /-/
 @ File @ Help
%+--------------------------------
<Command ===>_ZCMD                                              ~
<
<FileName . . [FILENAME                                        ~
<Member . . . [MEMNAME ~+
<
]PF1<Help ]PF3<End ]PF4<Prompt ]PF10<Actions
)INIT
&ZCMD = ''     /* clear command */
&ABCMD = ''    /* clear command */
&PDCMD = ''    /* clear AB command */
&SMSG = ''     /* clear short message */
&TEXT2 = '        '
PANEXIT ((FILENAME,TEXT2),REXX,XPXCMD1)

)REINIT
&ZCMD = ''     /* clear command */
REFRESH(*)

)PROC
IF (.RESP = END)
  EXIT

/* formally validate <filename> */
VER(&FILENAME,NB,DSNAME)
IF (.MSG NE '') EXIT
VPUT (FILENAME) PROFILE

IF (&ZCMD = PROMPT)
  &CURFLD = .CURSOR
  &ZWINTTL = TRANS(&CURFLD
                   MEMNAME,'Select Member Name'
                   MSG=XXMSGØØ3)
  &ZCMD = '&CURFLD = PCMD("&CURFLD",&FILENAME)'
  VPUT (ZWINTTL)
  EXIT

IF (&ABCMD NE '')
  EXIT

/* validate the existence of the file */
&MSG = '                         ' /* clear 32 bytes */
PANEXIT ((FILENAME,MSG),REXX,XPXCMD2)
IF (&MSG NE ' ')
  &LMSG = &MSG
  .MSG = XXMSGØØ5
```

```
  EXIT

/* formally validate member name */
VER(&MEMNAME,NAME) /* verify whether <memname> is valid NAME */
IF (&MEMNAME EQ '') /* check whether <memname> specified */
  &LMSG = 'Enter or prompt for member name'
  .MSG = XXMSGØØ5

)END
```

DEMOPAN, the display panel, has the following form when it is displayed:

```
+-----------------------------------------------------------------+
¦------------------------ Edit Frontend --------------------------¦
¦   File   Help                                                   ¦
¦ ------------------------------------------------------------- ¦
¦ Command ===>                                                   ¦
¦                                                                ¦
¦ FileName . .                                                   ¦
¦ Member . . .           +                                       ¦
¦                                                                ¦
¦ PF1 Help  PF3 End  PF4 Prompt  PF1Ø Actions                    ¦
+-----------------------------------------------------------------+
```

The **File** pull-down menu has the following form when it is displayed:

```
+--------+
¦New...  ¦
¦*Open...¦
+--------+
```

An * is prefixed to Open when the associated dataset does not have any members; the XPXCMD1 panel exit sets the TEXT2 variable appropriately. Although the Help action bar has two entries, to avoid overcomplicating the example, it serves only as a placeholder here.

Depending on the entry selected in the **File** pull-down menu, either the XACMD command is invoked directly (for New) or the ABCMD variable is set to invoke the XBCMD (for Old) procedure at the end of the panel display.

The XACMD command specified in the command table (‘SELECT CMD(ACMD &ZPARM)’) specifies that the ACMD procedure is invoked. ACMD allocates the specified dataset.

The DEMOCMDS command table assigns the following commands with the specified action:

```
ZCTVERB = 'PROMPT'
ZCTACT = 'PASSTHRU'
ZCTVERB = 'XACMD'
ZCTACT = 'SELECT CMD(ACMD &ZPARM)'
```

The MEMBER input field in the panel is a prompt field. The DEMOKEYS keylist table assigns the following commands for the DISPLAY keylist name. The KEYLIST parameter in the *)PANEL* header, KEYLIST(DISPLAY,DEMO), specifies the DISPLAY entry in the DEMO keylist, namely DEMOKEYS:

```
PF1 = HELP
PF3 = END
PF4 = PROMPT
PF10 = ACTIONS
All other PF-keys are disabled (set to NOP).
```

XBCMD (LIST LIBRARY AND SELECT MEMBER)

```
/* REXX - XBCMD */
/* Task: List library and select member */
PARSE ARG libname
ADDRESS ISPEXEC
"LMINIT DATAID(did) DATASET("libname")"
IF RC <> 0 THEN DO
  SAY "Open error: "libname
  EXIT ''
END
"LMOPEN DATAID("did") OPTION(INPUT)"

"SETMSG MSG(XXMSG006)" /* MSG: select member */
"LMMDISP DATAID("did") OPTION(DISPLAY) COMMANDS(S)"
IF RC = 0
  THEN member = zlmember
  ELSE DO
    member = ''
    "SETMSG MSG(XXMSG007)" /* MSG: no member selected */
  END
"LMCLOSE DATAID("did")"
RETURN member
```

The XBCMD procedure is invoked when **File/Old** is selected. The procedure uses the LMMDISP service to display the list of members in the associated library. The 'S' command can be used to select a member, the name of which is returned as a procedure result.

PCMD (PROMPT FOR MEMBER)

```
/* REXX - PCMD */
/* Task: Prompt for member */
PARSE ARG fldname,libname
ADDRESS ISPEXEC
/* test whether empty library */
rc = LISTDSI(libname "DIRECTORY")
IF RC > 8 THEN DO
  SAY "Open error: "libname
  EXIT ''
END
IF sysmembers = Ø THEN DO
  SAY "Empty library"
  member = ''
  DO WHILE member = ''
    SAY "Enter new member name"
    PULL member
    member = LEFT(member,8)
  END
  RETURN member
END
/* otherwise select existing member */
"SETMSG MSG(XXMSGØØ6)" /* MSG: select member */
"LMINIT DATAID(did) DATASET("libname")"
"LMOPEN DATAID("did") OPTION(INPUT)"

"ADDPOP"
"LMMDISP DATAID("did") OPTION(DISPLAY) COMMANDS(S)"
IF RC = Ø  THEN member = zlmember  ELSE DO    member = ''    "SETMSG
MSG(XXMSGØØ8)" /* MSG: no member selected */
  END
"REMPOP""LMCLOSE DATAID("did")"
RETURN member
```

The PCMD procedure is invoked as a result of pressing the PF4 key when the cursor is positioned on the MEMBER field. The user is requested to enter a member name if the associated library does not contain any members, otherwise the list of members in the associated library is displayed for selection. The procedure returns the member name (either new or selected) as a result.

The TRANS() function executed in the panel processing section when the ZCMD variable contains PROMPT (namely the PF4 key has been pressed) performs two tasks:

1 Tests whether CURFLD (ie the current cursor location) is MEMNAME. The message XXMSG003 will be set if this is not the case, ie the cursor is not placed at a prompt field.

2 Assigns the text ‘Select Member Name’ as a window title for the associated display.

XPXCMD1 (PANEL EXIT TO TEST WHETHER FILE HAS ANY MEMBERS)

```
/* REXX - XPXCMD1 panel exit1 */ /* Task: Test whether file has any
members */
PARSE ARG parmCALL ISPREXPX('I') /* set local REXX variables */
/* varnames.1: FILENAME */
/* varnames.2: TEXT2 */
text2 = LEFT('*Open...',varlens.2)
txt = SYSDSN(filename)
IF txt = 'OK' THEN DO
  CALL LISTDSI filename 'DIRECTORY'
  IF sysdsorg = PO & sysmembers <> Ø THEN,
    text2 = LEFT('Open...',varlens.2) /* set message text */
END
CALL ISPREXPX('T') /* set ISPF variables */
```

XPXCMD2 PANEL EXIT TO TEST FOR THE EXISTENCE OF THE SPECIFIED FILENAME)

```
/* REXX - XPXCMD2 panel exit2 */
/* Task: Test existence of filename */
PARSE ARG parm
CALL ISPREXPX('I') /* set local REXX variables */
/* varnames.1: FILENAME */
/* varnames.2: MSG */
txt = SYSDSN(filename)
IF txt <> 'OK',
  THEN msg = LEFT(txt,varlens.2) /* set message text */
CALL ISPREXPX('T') /* set ISPF variables */
```

ACMD (ALLOCATE DATASET)

```
/* REXX - ACMD */
/* Task: Allocate dataset (simplified) */
ARG dsns
ADDRESS ISPEXEC
DO FOREVER
  "DISPLAY PANEL(ISRUAASE)"
  IF RC = Ø THEN DO
    recfm = ''
    DO WHILE zalrf <> ''
      PARSE VAR zalrf temp 2 zalrf
      recfm = recfm temp
```

```
    END
    IF zalspac = 'TRKS' THEN zalspac = 'TRACKS'
    IF zalspac = 'CYLS' THEN zalspac = 'CYLINDERS'
    allocstr = ,
    "ALLOC F(DD) DSN("dsns") SPACE("zal1ex")" zalspac,
    "BLKSIZE("zalblk") LRECL("zallrec") RECFM("recfm")",
    "NEW REUS"
    IF zaldir <> Ø THEN allocstr = allocstr "DIR("zaldir")"
    ADDRESS TSO allocstr
    SAY "ALLOC RC:"rc
    IF RC = Ø THEN LEAVE
  END
  ELSE LEAVE
END
```

DEMOCMDS (COMMAND TABLE)

```
/* REXX - DEMOCMDS */
isptabl = 'demo.tlib'
SAY "COMMANDS being added"
ADDRESS TSO "ALLOC F(ISPTABL) DA("ISPTABL") SHR REUS"
ADDRESS ISPEXEC
"TBCREATE DEMOCMDS NAMES(ZCTVERB ZCTTRUNC ZCTACT ZCTDESC) REPLACE"
IF RC > 4 THEN DO
  SAY "DEMOCMDS cannot be created"
  EXIT
END
SAY TBCREATE RC
ZTDESC = ''
ZCTVERB = 'PROMPT'
ZCTTRUNC = Ø
ZCTACT = 'PASSTHRU'
"TBADD DEMOCMDS"
SAY TBADD RC
ZCTVERB = 'XACMD'
ZCTTRUNC = Ø
ZCTACT = 'SELECT CMD(ACMD &ZPARM)'
"TBADD DEMOCMDS"
SAY TBADD RC
"TBSAVE DEMOCMDS"
SAY TBSAVE RC
"TBCLOSE DEMOCMDS"
IF RC > 4 THEN DO
  SAY "DEMOCMDS error" RC
  EXIT
END
SAY TBCLOSE RC
```

The example assumes that the DEMOCMDS command table is stored

as DEMOCMDS, a member in the user's DEMO.TLIB library. This library must be contained in the ISPTLIB concatenation at runtime.

DEMOKEYS (KEYLIST TABLE)

```
/* REXX - DEMOKEYS */
isptabl = 'demo.tlib'
SAY "KEYS being added"
ADDRESS TSO "ALLOC F(ISPTABL) DA("isptabl") SHR REUS"
ADDRESS ISPEXEC
"TBCREATE DEMOKEYS KEYS(KEYLISTN) ",
 " NAMES(KEY1DEF KEY1LAB KEY1ATR ",
 "       KEY2DEF KEY2LAB KEY2ATR ",
 "       KEY3DEF KEY3LAB KEY3ATR ",
 "       KEY4DEF KEY4LAB KEY4ATR ",
 "       KEY5DEF KEY5LAB KEY5ATR ",
 "       KEY6DEF KEY6LAB KEY6ATR ",
 "       KEY7DEF KEY7LAB KEY7ATR ",
 "       KEY8DEF KEY8LAB KEY8ATR ",
 "       KEY9DEF KEY9LAB KEY9ATR ",
 "       KEY1ØDEF KEY1ØLAB KEY1ØATR ",
 "       KEY11DEF KEY11LAB KEY11ATR ",
 "       KEY12DEF KEY12LAB KEY12ATR) ",
 " REPLACE"
SAY TBCREATE RC
KEYLISTN = DISPLAY
KEY1DEF = HELP
KEY1LAB = HELP
KEY1ATR = YES
KEY2DEF = NOP
KEY2LAB = NOP
KEY2ATR = NO
KEY3DEF = END
KEY3LAB = END
KEY3ATR = NO
KEY4DEF = PROMPT
KEY4LAB = PROMPT
KEY4ATR = YES
KEY5DEF = NOP
KEY5LAB = NOP
KEY5ATR = NO
KEY6DEF = NOP
KEY6LAB = NOP
KEY6ATR = NO
KEY7DEF = NOP
KEY7LAB = NOP
KEY7ATR = NO
KEY8DEF = NOP
KEY8LAB = NOP
KEY8ATR = NO
```

```
KEY9DEF = NOP
KEY9LAB = NOP
KEY9ATR = NO
KEY1ØDEF = ACTIONS
KEY1ØLAB = ACTIONS
KEY1ØATR = YES
KEY11DEF = NOP
KEY11LAB = NOP
KEY11ATR = NO
KEY12DEF = NOP
KEY12LAB = NOP
KEY12ATR = NO
"TBADD DEMOKEYS"
SAY TBADD RC
KEYLISTN = DATA
KEY1DEF = HELP
KEY1LAB = HELP
KEY1ATR = NO
KEY3DEF = END
KEY3LAB = END
KEY3ATR = NO
KEY4DEF = NOP
KEY4LAB = NOP
KEY4ATR = NO
KEY1ØDEF = NOP
KEY1ØLAB = NOP
KEY1ØATR = NO
"TBADD DEMOKEYS"
SAY TBADD RC
"TBSAVE DEMOKEYS"
SAY TBSAVE RC
"TBCLOSE DEMOKEYS"
SAY TBSAVE RC
```

This keylist defines two entries – DISPLAY and DATA (although the DATA is not used in this example). The example assumes that the keylist is stored as DEMOKEYS, a member in the user's DEMO.TLIB library. This library must be contained in the ISPTLIB concatenation.

XXMSG00 (MESSAGE MEMBER)

```
XXMSGØØ3 'Not a prompt field' .ALARM=YES
'The cursor has been placed on a field that does not provide prompt
capability'
XXMSGØØ5 '&SMSG' .ALARM=NO .WINDOW=NORESP
'&LMSG'
XXMSGØØ6 'SELECT MEMBER' .ALARM=YES
'ENTER "S" TO SELECT MEMBER'
XXMSGØØ7 'NO MEMBER SELECTED' .ALARM=YES
```

```
'EITHER NO MEMBER SELECTED OR EMPTY DATASET'
XXMSGØØ8 'NO MEMBER SELECTED' .ALARM=YES
''
```

APPENDIX

Rather than using the ISPREXPX() function, the standard REXX STORAGE() function can be used to access and set the passed ISPF variables. The example makes it clear that this is more involved.

XPXCMD2A

XPXCMD2A is an alternative version of XPXCMD2 using the STORAGE instruction to access and set the passed ISPF variables.

```
/* REXX - XPXCMD2A panel exit2 */
/* Task: Test existence of filename */
/* Access passed ISPF variables using the STORAGE function */
/* vn.1: FILENAME */
/* vn.2: MSG (set if error) */
PARSE ARG parm /* <parm> contains the address of the parameter list */
px = STORAGE(parm,32) /* <px> = parameter list */
/* Set <nv> = number of variables */
p5 = substr(px,17,4)
nv = C2X(STORAGE(C2X(p5),4))
/* Extract individual variable data lengths */
p7 = substr(px,25,4)
vla = STORAGE(C2X(p7),nv*4) /* variable length array */
vla = C2X(vla)
vl. = ,'
DO i = 1 TO nv
  x = SUBSTR(vla,i*8-7,8)
  vl.i = X2D(x)
END
/* Accumulate total length */
totvl = Ø
DO i = 1 TO nv
  totvl = totvl + vl.i
END
/* Get address and value of variables */
p8 = substr(px,29,4)
va = C2X(p8) /* variable address */
vva = STORAGE(va,totvl) /* variable value array */
va. = ,'
vv. = ,'
j = 1
DO i = 1 TO nv
  va.i = va
```

```
  vv.i = SUBSTR(vva,j,vl.i) /* extract individual values */
  j = j + vl.i
  x = X2D(va)
  x = x + vl.i
  va = D2X(x)
END
/* Set <filename> */
filename = vv.1
txt = SYSDSN(filename)
IF txt <> ,OK' THEN DO
  msg = LEFT(txt,vl.2) /* set message text */
  CALL STORAGE va.2,vl.2,txt
END
```

Systems Programmer
(Germany)

The fastest way to get SMS DASD space information

INTRODUCTION

As installations get bigger, it's essential to use DFSMS as an indispensable and handy storage management tool. DFSMS does its job very well, and tools like ISMF can supply all the necessary information in a quick and versatile way. But, unfortunately, they do not always cover every need – for example the console operator might need to know how the pools are or something similar, and sometimes it's necessary to monitor the available space throughout the day, or perhaps we need to have reports about the occupation of our disks.

For these cases, solutions like DCOLLECT, ISMF/BATCH, or directly using LISTVTOC are usually applied, but they are inefficient when we have a large number of disks at an installation. They're tedious to use because we have to add steps that will analyse the information or, on the other hand, we can't run them online because of system restrictions.

After testing some alternative methods I have reached the conclusion that the fastest, cleanest, and smartest way to get results is by using DFSMS directly.

THE SMS SUBSYSTEM INTERFACE

The DFSMS tool works on an MVS system through the little-known and worse-documented Subsystem Interface. Every operation related to files in an SMS environment goes through an internal call to the Subsystem Interface, from the simple creation of a small file to the activation of a new configuration.

Between the multiple calls available in the system, there are five functions that will serve our needs pretty well. With them we will be able to obtain the necessary information about volumes or storage groups.

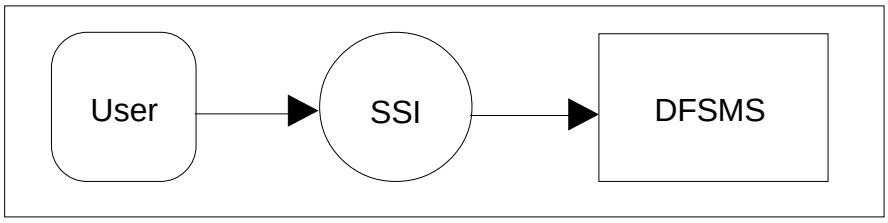

Figure 1: DFSMS request using SSI

We can accomplish the DFSMS request as shown in Figure 1.

It's necessary to create two control blocks, SSOB and SSSA, as well as to GETMAIN a buffer area to get the requested information. We will obtain the subsystem identification block (SSIB) related to DFSMS from the JES2 control blocks. As you can see in Figure 2, we have established the environment and, with the required control blocks created and available, the request is made calling the IEFSSREQ macro.

The subfunction codes that we are interested in are:

- SSSA1SGV – return volume record definitions from a given SG.
- SSSA1VSG – return an SG from a given volser.
- SSSA1VOL – return a volume record definition from a given

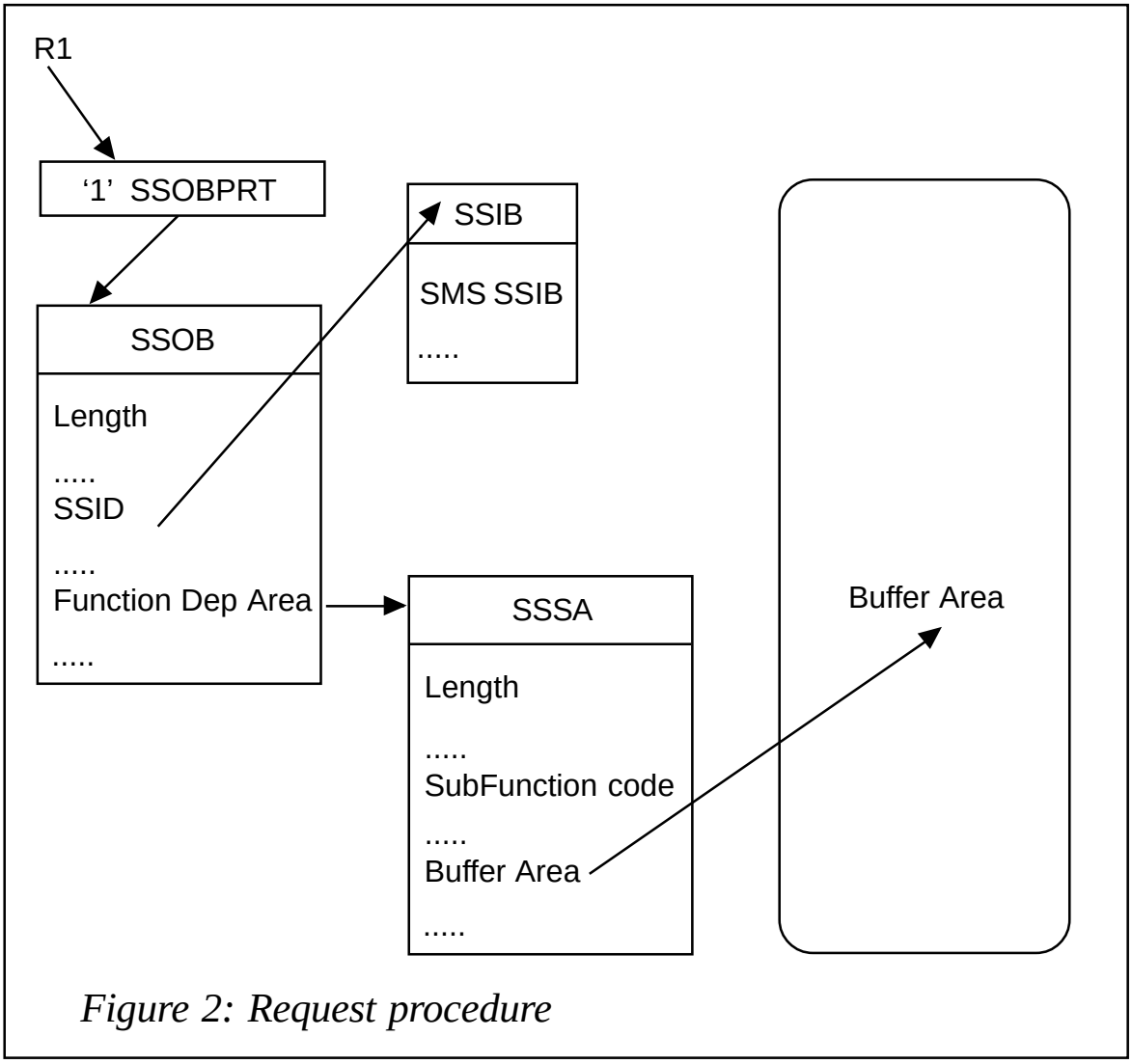

Figure 2: Request procedure

volser.

- SSSA1AVL – return all volume record definitions in the current configuration.
- SSSA1SGL – return all storage group definitions in the current configuration.

In any case, DFSMS provides a structure in the buffer area, for the requested information, which can be mapped to the IGDSGD and IGDVLD macros. This information is retained in main memory through DFSMS, making the access to any VTOC unnecessary – so that's the trick of this quick method.

QUERYSMS REXX FUNCTION

I decided to write a function in REXX, as opposed to a program, to make better use of the precious information that the interface can give us.

I've always been an ASM 370/390 lover, but I think that you should only write your program in Assembler language if you can't solve the problem more easily with a high-level language.

In every situation, I write the main function in ASM code and one or more REXX routines that use it, so I can play with every possible way of making the critical code shorter.

The QUERYSMS function accepts two values – kind of call (SGV/VSG/VOL/AVL/SGL), and one name (where required).

The reply is put back to the stack where, once formatted, it can be used for virtually any need. In the example REXX program, two routines are included to format both registers: VLD (Volume Definition Record) and SGD (Storage Group Definition).

DFSMS can return enough information about volumes and SGs (capacity, free space, status), so we can use a large range of utilities to produce reports with virtually any information.

However, the program nucleus is self-sufficient from the REXX environment, as it can be converted to a batch program or TSO command with very little effort.

BUILDING AND INSTALLING

Before we proceed to the compilation, the internal parameter of the program MAXVOLS (line 2) must be adjusted, indicating the maximum number of SMS volumes that the installation owns. This is a critical and important adjustment, so, if we don't define a buffer with enough space available, we will not be able to use this function to list all the volumes (AVL). It must be greater than the real number of volumes, because the SMS uses approximately 2K for every volser.

The QUERYSMS module doesn't require any special compilation parameter, except for the concatenation of the SYS1.MODGEN macros library in the assemble step.

Once constructed, it must reside in any standard 'system search order'

load library, LINKLST, STEPLIB, or ISPLLIB.

AN EXAMPLE

The VIEWSGS REXX program is a good example of the use of the QUERYSMS function. The program calls the function to get a full list of all storage groups in the active configuration (Parm SGL), then calls the Format_SGD routine (which set ups the REXX variables with the SG settings), and then we keep in one table those that we need in order to list all the volumes for each of them (Parm SGV). Now, we call the Format_VLD routine to get the free space and capacity on every volume. They build up in counters and finish displaying the whole SG.

At a typical medium-size installation with approximately a thousand disks, the whole process takes less than a second.

Output example:

```
 Storage Group Name     Volumes  Capacity      Free  %Used
-------------------------------------------------------------------------------
CATALOGS                    15     41565     23786  42,77 =========..........
DATABASES                  340    942140    124559  86,78 =================...
LIBRARY                     20     55420     35234  36,42 =======.............
PRIMARY                    235    651185    230568  64,59 =============.......
TEMP                        55    152405     15650  89,73 ==================..
TSOUSERS                    15     41565     33687  18,95 ====................
```

QUERYSMS

```
QUERYSMS TITLE 'REXX FUNCTION - QUERY SMS'
&MAXVOLS SETA  2000 /* MAX DASD SMS VOLUMES IN THE SYSTEM */
*********************************************************************
*             MODULE NAME = QUERYSMS                                *
*             DESCRIPTIVE NAME =  REXX FUNCTION TO QUERY SMS THROW SSI*
*             AUTHOR = SALVADOR CARRASCO                            *
*             FUNCTION =                                            *
*                arg(1)='SGV' - return a list of volumes associated   *
*                               the storage group arg(2)              *
*                       'VSG' - return the storage group associated   *
*                               the volume arg(2)                     *
*                       'VOL' - return arg(2) volume definition       *
*                       'AVL' - return the list of all volumes        *
*                       'SGL' - return the list of storage groups     *
*             RESTRICTIONS = NONE                                   *
*             REGISTER CONVENTIONS = STANDARD CONVENTIONS.          *
*             ATTRIBUTES = KEY NZ, PROBLEM STATE, ENABLED, NO LOCKS   *
```

```
*             ENTRY POINTS = QUERYSMS (ONLY ENTRY POINT)              *
*             INPUT = REGØ POINTS TO REXX ENVBLOCK                    *
*                     REG1 POINTS TO REXX ARGTABLE                    *
*             EXIT - NORMAL = AT PROGRAM END VIA BR 14                *
*             OUTPUT = DATA IN REXX STACK                             *
*                      RETURN CODE = 1                                *
*             EXIT - ERROR = AT PROGRAM END VIA BR 14                 *
*             OUTPUT = ERROR MSG TO IRXSAY                            *
*                      RETURN CODE = 16                               *
*             EXTERNAL REFERENCES =                                   *
*                CONTROL BLOCKS = VARIOUS MVS/SMS.                    *
*                                 SEE BOTTOM OF CODE.                 *
************************************************************************
QUERYSMS CSECT
QUERYSMS AMODE 31
QUERYSMS RMODE ANY
         YREGS                          STD REGISTER EQUATES
* ADDRESSING
         SAVE  (14,12),,*               SAVE PREVIOUS
         LR    R12,R15                  BASE REG
         USING QUERYSMS,R12             =BASE REGISTER     =
         LR    R11,RØ                   REXX ENVBLOCK ADD.
         LR    R1Ø,R1                   REXX EFPL ADD.
         GETMAIN RU,LV=LDATA,LOC=BELOW  GET WORK AREA
         LR    R15,R13                  PREVIOUS SAVEAREA
         LR    R13,R1                   NEW SAVEAREA/WORK AREA
         ST    R15,4(R13)               SAVE OLD SAVEAREA
         ST    R13,8(R15)               SAVE NEW SAVEAREA
         USING DATA,R13                 =SAVEAREA+WORK AREA=
         USING ENVBLOCK,R11             =REXX ENV BLOCK    =
         USING EFPL,R1Ø                 =REXX EFPL         =
         USING EVALBLOCK,R9             =REXX EVALBLOCK    =
         USING ARGTABLE_ENTRY,R8        =REXX ARG TABLE    =
         L     R9,EFPLEVAL              USE DEFAULT EVALBLOCK
         L     R9,Ø(R9)                 ADDRESS IT
         L     R8,EFPLARG               FIRST ARG
* GET AND VERIFY PARM
         XC    PARM1,PARM1              CLEAR PARM1
         MVC   PARM2,=CL3Ø' '           CLEAR PARM2
         L     R1,ARGTABLE_ARGSTRING_PTR  GET FIRST ARG ADD
         LTR   R1,R1                    TEST
         BM    EXITNOK                  NO PARM -> ERROR
         L     R2,ARGTABLE_ARGSTRING_LENGTH GET LENGTH OF ARG
         C     R2,=F'3'                 LENGTH = 3 ?
         BNE   EXITNOK                  NO -> EXIT NO OK
         BCTR  R2,Ø                     LEN=LEN-1
         EX    R2,COPYPAR1              EXECUTE MOVE
         LA    R8,ARGTABLE_NEXT         POINT TO NEXT ARG
         L     R1,ARGTABLE_ARGSTRING_PTR GET ARG ADD
         LTR   R1,R1                      TEST
```

```
         BM    CHECKP1                   NO PARM2 -> SKIP
         L     R2,ARGTABLE_ARGSTRING_LENGTH GET LENGTH OF ARG
         C     R2,=F'3Ø'                 LENGTH <= 3Ø ?
         BH    EXITNOK                   NO -> EXIT NO OK
         STH   R2,PARM2L                 SAVE LENGTH
         BCTR  R2,Ø                      LEN=LEN-1
         EX    R2,COPYPAR2               EXECUTE MOVE
CHECKP1  EQU   *
         MVI   TYPE,C'V'                 VLD PROCESS
         CLC   PARM1,=C'VOL'             ARG(1) = 'VOL' ?
         BE    CHECKP2                   YES, CHECK ARG(2)
         CLC   PARM1,=C'AVL'             ARG(1) = 'AVL' ?
         BE    MAIN                      YES, PROCEED
         CLC   PARM1,=C'SGV'             ARG(1) = 'SGV' ?
         BE    CHECKP2                   YES, PROCEED
         MVI   TYPE,C'S'                 SGD PROCESS
         CLC   PARM1,=C'VSG'             ARG(1) = 'VSG' ?
         BE    CHECKP2                   YES, PROCEED
         CLC   PARM1,=C'SGL'             ARG(1) = 'SGL' ?
         BE    MAIN                      YES, PROCEED
         B     EXITNOK                   ARG(1) NOT VALID
CHECKP2  CLC   PARM2,=CL3Ø' '            ARG(2) EMPTY ?
         BE    EXITNOK                   YES, EXIT NO OK
         B     MAIN                      ARG(2) OK -> PROCEED
COPYPAR1 MVC   PARM1(Ø),Ø(R1)            MOVE ARG(1)
COPYPAR2 MVC   PARM2(Ø),Ø(R1)            MOVE ARG(2)
* MAIN
MAIN     GETMAIN RU,LV=WKTBL,LOC=ANY     GET WORK AREA
         ST    R1,WORKVLD                SAVE BUFFER ADDRESS
         BAL   R14,CALLSMS               GET SMS INFORMATION
         BAL   R14,SAVEDAT               WRITE DATA
         L     R1,WORKVLD                GET BUFFER ADDRESS
         FREEMAIN RU,LV=WKTBL,A=(1)      FREE SAVE/WORK AREA
         B     EXITOK
* EXIT
EXITBAD1 CVD   R15,WORKD                 CONV TO DECIMAL SSI RC
         ED    ABEØ1M(6),WORKD+5         EDIT IN MSG
         LA    R1,ABEØ1                  LOAD MSG ADD
         BAL   R14,WRITEER               SAY ERROR
         LA    R11,16                    RETURN CODE=16
         B     $EØ2
EXITBAD2 CVD   R1,WORKD                  CONV TO DECIMAL SSSA RSN
         ED    ABEØ2M(6),WORKD+5         EDIT IN MSG
         LA    R1,ABEØ2                  LOAD MSG ADD
         BAL   R14,WRITEER               SAY ERROR
         LA    R11,16                    RETURN CODE=16
         B     $EØ2
EXITNOK  LA    R1,ERRØ1                  ARG/PARM ERROR
         BAL   R14,WRITEER               SAY ERROR
         LA    R11,16                    RC=16
```

```
         B     $EØ2
EXITOK   MVI   EVALBLOCK_EVDATA,C'1'   RETURN 'TRUE'
         LA    R1,1                    RETURN LENGTH
         ST    R1,EVALBLOCK_EVLEN      SAVE IT
         SLR   R11,R11                 RETURN CODE=Ø
$EØ2     LR    R1,R13                  GET SAVE/WORK AREA
         L     R13,4(13)               GET OLD SAVEAREA
         FREEMAIN RU,LV=LDATA,A=(1)    FREE SAVE/WORK AREA
         LR    R15,R11                 GET RETURN CODE
         RETURN (14,12),RC=(15)        RETURN TO CALLER
* CALLSMS
CALLSMS  EQU   *
         ST    R14,SCALLSMS            SAVE RETURN ADDRESS.
* CONSTRUCT SSOB
         XC    WORKSSOB,WORKSSOB       CLEAR SSOB
         LA    R2,WORKSSOB             LOAD SSOB ADD.
         USING SSOB,R2                 MAP IT
         MVC   SSOBID,=C'SSOB'         SSOB ACRON.
         MVC   SSOBLEN,=AL2(SSOBHSIZ)  SSOB LENGTH
         MVC   SSOBFUNC,=AL2(SSOBSSMS) FUNCTION CODE FOR SMS SERVICES
         L     R1,CVTPTR(Ø,Ø)          -> CVT
         L     R1,CVTJESCT-CVTMAP(,R1) -> JES2 COMMUNICATION TABLE
         L     R1,JESCTEXT-JESCT(,R1)  -> PAGEABLE JESCT
         MVC   SSOBSSIB,JESSMSIB-JESPEXT(R1) -> SMS SSIB
         O     R2,=X'8ØØØØØØØ'         SET LAST PARM
         ST    R2,PTRSSOB              SAVE IT
* CONSTRUCT SSSA
         XC    WORKSSSA,WORKSSSA       CLEAR SSSA
         LA    R3,WORKSSSA             LOAD SSSA ADD.
         ST    R3,SSOBINDV             FUNCTION DEPENDENT AREA POINTER
         USING IEFSSSA,R3              MAP IT
         MVC   SSSAID,=C'SSSA'         SSSA ACRON.
         LA    RØ,SSSALN+SSSA1LN+32    LENGTH
         STH   RØ,SSSALEN
         MVC   SSSAVER,=AL2(SSOBSSVR)  VERSION NUMBER
         MVC   SSSASFN,=AL2(SSSAACTV)  RETURNS DATA FROM THE ACTIVE CFG
         MVI   SSSAIFLG,SSSANAUT       CALLER NOT AUTHORIZED
$CØ      EQU   *
         CLC   PARM1,=C'AVL'           ARG(1) = 'AVL' ?
         BNE   $C1                     NO, TRY NEXT
         MVI   SSSA1TYP,SSSA1AVL       RETURN THE LIST OF ALL VOLUMES
         B     $C9
$C1      CLC   PARM1,=C'VOL'           ARG(1) = 'VOL' ?
         BNE   $C2                     NO, TRY NEXT
         MVI   SSSA1TYP,SSSA1VOL       RETURN A VOLUME
         MVC   SSSA1NML,=AL2(6)        LENGTH OF VOLSER
         MVC   SSSA1NAM(6),PARM2+Ø     MOVE VOLSER
         B     $C9
$C2      CLC   PARM1,=C'SGV'           ARG(1) = 'SGV' ?
         BNE   $C3                     NO, TRY NEXT
```

```
         MVI   SSSA1TYP,SSSA1SGV       RETURN A LISTVOL FROM A STORGRP
         MVC   SSSA1NML,PARM2L         LENGTH OF STORGRP NAME
         MVC   SSSA1NAM(3Ø),PARM2+Ø    MOVE STORGRP
         B     $C9
$C3      CLC   PARM1,=C'VSG'           ARG(1) = 'VSG' ?
         BNE   $C4                     NO, TRY NEXT
         MVI   SSSA1TYP,SSSA1VSG       RETURN THE SG FROM A VOLUME
         MVC   SSSA1NML,=AL2(6)        LENGTH OF VOLSER
         MVC   SSSA1NAM(6),PARM2+Ø     MOVE VOLSER
         B     $C9
$C4      EQU   *                       ARG(1) = 'SGL'
         MVI   SSSA1TYP,SSSA1SGL       RETURN SG LIST
         B     $C9
$C9      EQU   *
         MVC   SSSA1CNT,=F'1'          ONE REQUEST
         MVC   SSSA1LEN,=AL4(WKTBL)    LENGTH OF WORK AREA
         L     R1,WORKVLD              GET VLD WORK AREA
         ST    R1,SSSA1PTR             SAVE VLD WORK AREA
* CALL SSI
         LA    R1,PTRSSOB              GET ADD OF SSOB ADD
         IEFSSREQ                      CALL SSI
         LTR   R15,R15                 TEST RC
         BNZ   EXITBAD1                ¬=Ø -> EXIT BAD RETURN CODE
         L     R1,SSSARSN              LOAD SSSA REASON CODE
         LTR   R1,R1                   TEST RC
         BNZ   EXITBAD2                ¬=Ø -> EXIT BAD RETURN CODE
         DROP  R2,R3                   END OF PROCESS
RCALLSMS L     R14,SCALLSMS            GET RETURN ADD.
         BR    R14                     RETURN TO CALLER
* SAVEDAT - SAVE RETURNED INFORMATION
SAVEDAT  EQU   *
         ST    R14,SSAVEDAT            SAVE CALLER'S RET. ADD.
* PREPARE CALL TO QUEUE
         L     R1,ENVBLOCK_IRXEXTE     GET VECTOR EXTERNAL ROUTINES ADD
         L     R7,IRXSTK-IRXEXTE(,R1)  GET STACK ROUTINE ADD.
         ST    R11,_STKENV             SAVE ENV ADD
         MVC   _STKFUNC,=CL8'QUEUE'    SET QUEUE FUNCTION
         LA    R1,_STKFUNC             GET ADD
         ST    R1,_STKPARM+Ø           SET AS PARM1
         LA    R1,_STKDAT              GET DATA ADDRESS
         ST    R1,_STKPARM+4           SET AS PARM2
         LA    R1,_STKLEN              GET DATA LENGTH
         ST    R1,_STKPARM+8           SET AS PARM3
         LA    R1,_STKRC               GET RC AREA
         ST    R1,_STKPARM+12          SET AS PARM4
         LA    R1,_STKENV              GET ENVBLCOK ADD
         ST    R1,_STKPARM+16          SET AS PARM5
         LA    R1,_STKRCE              GET RCE ADDRESS
         O     R1,=X'8ØØØØØØØ'         SET AS LAST PARM
         ST    R1,_STKPARM+2Ø          SET AS PARM6
```

```
* WRITE ITEMS
         L     R2,WORKVLD              GET WORK ADD
         USING VLD,R2                  MAP AS VLD/SGD
         L     R3,VLDPCNT              GET TOTAL COUNT
         L     R4,VLDPLEN              GET LENGTH OF EACH ITEM
         ST    R4,_STKLEN              SET LENGTH FOR IRXSTK
         LA    R2,VLDEF                GET FIRST ENTRY
         DROP  R2
NEXT     ST    R2,_STKDAT              SET DATA ADDRESS
         LR    RØ,R11                  GET ENVBLOCK
         LR    R15,R7                  GET IRXSTK ENTRY POINT
         LA    R1,_STKPARM             LOAD PARM ADD
         BALR  R14,R15                 CALL IRXSTK
         AR    R2,R4                   SKIP TO NEXT ENTRY
         BCT   R3,NEXT                 REPEAT, UNTIL COUNT = Ø
RSAVEDAT L     R14,SSAVEDAT            GET CALLER RETURN ADD
         BR    R14                     RETURN TO CALLER
* WRITER - WRITE ERROR MSG
WRITEER  EQU   *
         ST    R14,SWRITEER
* PREPARE CALL TO SAY
         L     R2,ENVBLOCK_IRXEXTE     GET VECTOR EXTERNAL ROUTINES ADD
         L     R7,IRXSAY-IRXEXTE(,R2)  GET STACK ROUTINE ADD.
         ST    R11,_STKENV             SAVE ENV ADD
         MVC   _STKFUNC,=CL8'WRITEERR' SET WRITEERR FUNCTION
         LA    R2,_STKFUNC             GET ADD
         ST    R2,_STKPARM+Ø           SET AS PARM1
         LA    R2,_STKDAT              GET DATA ADDRESS
         ST    R2,_STKPARM+4           SET AS PARM2
         LA    R2,_STKLEN              GET DATA LENGTH
         ST    R2,_STKPARM+8           SET AS PARM3
         LA    R2,_STKENV              GET ENVBLCOK ADD
         ST    R2,_STKPARM+12          SET AS PARM4
         LA    R2,_STKRCE              GET RCE ADDRESS
         O     R2,=X'8ØØØØØØØ'         SET AS LAST PARM
         ST    R2,_STKPARM+16          SET AS PARM5
* WRITE ERROR MSG
         LR    R3,R1                   LOAD PARM ADD
         SR    R2,R2                   CLEAR R2
NEXLINE  ICM   R2,B'ØØ11',Ø(R3)        GET LINE LENGTH
         LTR   R2,R2                   IS ZERO ?
         BZ    RWRITEER                YES, -> END
         LA    R3,2(R3)                GET DATA ADDRESS
         ST    R3,_STKDAT              SET DATA TO SAY
         ST    R2,_STKLEN              SET LENGTH TO SAY
         LR    RØ,R11                  GET ENVBLOCK
         LR    R15,R7                  GET IRXSAY ENTRY POINT
         LA    R1,_STKPARM             LOAD PARM ADD
         BALR  R14,R15                 CALL IRXSAY
         AR    R3,R2                   ADD LINE LENGTH
```

```
         B     NEXLINE                 GO FOR NEXT LINE
RWRITEER L     R14,SWRITEER            GET RETURN ADD
         BR    R14                     RETURN TO CALLER
* DATA AREA
ERRØ1    DC    AL2(26),C'Usage: QUERYSMS(type,name)'
         DC    AL2(13),C' Where type ='
         DC    AL2(73),C'  SGV - return a list of volumes associated wiX
               th the storage group ''name'''
         DC    AL2(66),C'  VSG - return the storage group associated wiX
               th the volume ''name'''
         DC    AL2(39),C'  VOL - return ''name'' volume definition'
         DC    AL2(38),C'  AVL - return the list of all volumes'
         DC    AL2(41),C'  SGL - return the list of storage groups'
         DC    AL2(Ø)
ABEØ1    DC    AL2(39),C'Subsystem Interface Error, RC = '
ABEØ1M   DC    X'40202020212040'
         DC    AL2(33),C'  Note: See SYS1.MACLIB(IEFSSOBH)'
         DC    AL2(ØØ)
ABEØ2    DC    AL2(24),C'SMS Error, RSN = '
ABEØ2M   DC    X'40202020212040'
         DC    AL2(32),C'  Note: See SYS1.MODGEN(IEFSSSA)'
         DC    AL2(ØØ)
         LTORG
DATA     DSECT                         SAVE/WORK AREA
SAVEAREA DS    18F                     SAVE AREA
SSAVEDAT DS    F                       ROUTINE RETURN
SCALLSMS DS    F                       ROUTINE RETURN
SWRITEER DS    F                       ROUTINE RETURN
         DS    ØD                      ALIGN.
_STKFUNC DS    CL8                     STKFUNC
_STKDAT  DS    F                       STKDAT
_STKLEN  DS    F                       STKLEN
_STKRC   DS    F                       STKRC
_STKENV  DS    F                       STKENV
_STKRCE  DS    F                       STKRCE
_STKPARM DS    6F                      PARM ADDRESS
WORKD    DS    D                       WORKING DECIMAL
PARM1    DS    CL3                     PARM 1
PARM2    DS    CL3Ø                    PARM 2
PARM2L   DS    H                       PARM 2 LENGTH
TYPE     DS    X                       TYPE OF RETURN
PTRSSOB  DS    F                       -> TO SSOB
         DS    ØD                      ALIGN.
WORKSSOB DS    XL(SSOBHSIZ)            WORKING SSOB
         DS    ØD                      ALIGN.
WORKSSSA DS    XL(SSSALN+SSSA1LN+32)   WORKING SSSA
VLDLEN   EQU   VLDEND-VLDEF+VLDEND2-VLDSYSDT TOTAL VLD LENGTH
WKTBL    EQU   VLDEF-VLD+(VLDLEN*&MAXVOLS)
WORKVLD  DS    F
LDATA    EQU   *-DATA
```

```
* USED MAPS
DUMMY    DSECT
         IRXENVB                        REXX ENV. BLOCK
         IRXEFPL                        REXX EFPL
         IRXARGTB                       REXX ARG. TABLE
         IRXEVALB                       REXX EVAL BLOCK
         IRXEXTE                        REXX ROUTINES
         IEFUCBOB DEVCLAS=DA,PREFIX=YES  UCB MAPPING
         IEFJSSOB
         IEFSSSA                        SMS - SSI
         CVT DSECT=YES                  CVT
         IEFJESCT                       JESCT
         IGDVLD                         SMS VOLUME DEFINITION MAPPING
         IGDSGD                         SMS STORAGE GROUP DEFINITION MAP
         END   QUERYSMS
```

VIEWSGS

```
/*-------------------------------------------------------------Rexx--*/
/* VIEWSGS: View Space available in all Storage Groups               */
if querysms("SGL") then do
   SG_Count = queued()
   do i=1 to SG_Count
      parse pull sgd
      call Format_SGD sgd
      SG_Name.i = SGD_Sgdfname
      SG_Desc.i = SGD_Sgdfdesc
      SG_Type.i = SGD_Sgdftype
      end
   say left("Storage Group Name",2Ø)right("Volumes",1Ø)||,
       right("Capacity",1Ø)right("Free",1Ø)right("%Used",7)
   say copies("-",79)
   do i=1 to SG_Count
      if SG_Type.i = Ø then do
         vols = Ø; tc = Ø; tf = Ø
         if querysms("SGV",SG_Name.i) then do
            vols = queued()
            do j=1 to vols
               parse pull vld
               call Format_Vld vld
               tc = tc + VLD_Vldntcpy
               tf = tf + VLD_Vldnfree
               end
            end
         rt = format(((tc-tf)/tc)*2Ø,,Ø)
         say Left(SG_Name.i,2Ø)right(vols,1Ø)right(tc,1Ø)right(tf,1Ø)||,
             right(format(((tc-tf)/tc)*1ØØ,,2),7)" "copies("=",rt)||,
             copies(".",2Ø-rt)
         end
      end
```

```
   end
exit
/* Format VOLUME RECORD DEFINITION                                    */
Format_VLD:
parse arg 1   VLD_Vldvslen,          /* Volser length = 6             */
          3   VLD_Vldvser,           /* Volser                        */
          9   VLD_Reserved3,         /* Reserved                      */
          33  VLD_Vldduser,          /* userid of last updater        */
          41  VLD_Vlddate,           /* date of last update           */
          51  VLD_Vldtrksz,          /* Volume R1 track capacity      */
          53  VLD_Reserved4,         /* Reserved                      */
          57  VLD_Vlddtime,          /* Time of last update           */
          65  VLD_Vldsgl,            /* Length of storgrp name        */
          67  VLD_Vldsgn,            /* Name of storgrp               */
          97  VLD_Vldnstat,          /* Old status by system          */
          113 VLD_Vldnucba,          /* address of ucb if known       */
          117 VLD_Vldntcpy,          /* total capacity in megabytes   */
          121 VLD_Vldnfree,          /* amount free in megabytes      */
          125 VLD_Vldnlext,          /* largest free extent in mb.    */
          129 VLD_Vldflags,          /* Flags, see below              */
          130 VLD_Reserved5,         /* Reserved                      */
          131 VLD_Vldn0cnt,          /* Volume level Reset Count      */
          133 VLD_Vldnssi1,          /* Reserved for subsystem use    */
          137 VLD_Vldsgst,           /* storgrp status on this sytem  */
          138 VLD_Reserved5,         /* Reserved                      */
          141 VLD_Vldnlevl,          /* Update level for volume       */
          145 VLD_Vldcsmss,          /* Old location of confirmed stat*/
          153 VLD_Vldsysof,          /* Offset of system data         */
          157 VLD_Vldsysln,          /* Length of system data         */
          161 VLD_Reserved6,         /* Reserved                      */
          177 VLD_SistSTAT           /* Sistems STAT tables           */
VLD_Vldvslen = c2d(VLD_Vldvslen)
VLD_Vldtrksz = c2d(VLD_Vldtrksz)
VLD_Vldsgl = c2d(VLD_Vldsgl)
VLD_Vldsgn = substr(VLD_Vldsgn,1,VLD_Vldsgl)
VLD_Vldntcpy = c2d(VLD_Vldntcpy)
VLD_Vldnfree = c2d(VLD_Vldnfree)
VLD_Vldnlext = c2d(VLD_Vldnlext)
                                   /* Vol is in conversion         */
VLD_Flg_Conv = bitand(VLD_Vldflags,x'80') = x'80'
VLD_Vldn0cnt = c2d(VLD_Vldn0cnt)
VLD_Vldnlevl = c2d(VLD_Vldnlevl)
VLD_SmsStat. = ""                   /* SMS Status */
                                    /* 0 - No status given */
                                    /* 1 - Enabled */
                                    /* 2 - Quiesce/All */
                                    /* 3 - Quiesce/New */
                                    /* 4 - Disabled/All */
                                    /* 5 - Disabled/New */
VLD_MvsStat. = ""                   /* MVS Status */
                                    /* 1 - Online */
```

```
                                    /* 2 - Offline */
                                    /* 3 - Pending offline */
                                    /* 4 - Boxed */
                                    /* 5 - Not Ready */
VLD_SmsCStat. = ""                  /* Confirmed SMS Status */
                                    /* Same as SmsStat */
do vld_j = 1 to 8 /* Max systems 256 */
   VLD_SmsStat.vld_j = c2d(substr(VLD_SistSTAT,((vld_j-1)*8)+1,1))
   VLD_MvsStat.vld_j = c2d(substr(VLD_SistSTAT,((vld_j-1)*8)+2,1))
   VLD_SmsCStat.vld_j = c2d(substr(VLD_SistSTAT,((vld_j-1)*8)+3,1))
   end
return
/* Format STORAGE GROUPS RECORD DEFINITION                          */
Format_SGD:
parse arg 1   SGD_Sgdnmlen,         /* Reserved (would be name len)  */
          3   SGD_Sgdfname,         /* Storage Group Name            */
          33  SGD_Sgdfuser,         /* USERID of last updater        */
          41  SGD_Sgdfdate,         /* Date last updated             */
          51  SGD_Reserved1,        /* Reserved                      */
          57  SGD_Sgdftime,         /* Time last updated             */
          65  SGD_Sgdfdesc,         /* Description of Storage Group  */
          185 SGD_Sgdflags,         /* Flags and reserved, see below */
          186 SGD_Sgdftype,         /* Storage Group Type See below  */
          187 SGD_Reserved2,        /* Reserved                      */
          189 SGD_Sgdfvmax,         /* VIO MAX Data set size         */
          193 SGD_Sgdfvunt,         /* VIO unit type                 */
          197 SGD_Sgdfhthr,         /* High threshold Ø TO 99 %      */
          198 SGD_Sgdflthr,         /* Low threshold Ø TO 99 %       */
          199 SGD_Sgddmpcl,         /* Dump Classes for autodump     */
          239 SGD_Sgdfprst,         /* Old location: processor status*/
          247 SGD_Sgdabsys,         /* Auto backup system            */
          255 SGD_Sgdadsys,         /* Auto dump system              */
          263 SGD_Sgdamsys,         /* Auto migrate system           */
          271 SGD_Reserved2,        /* Reserved                      */
          273 SGD_Sgdssi1,          /* Reserved for subsystem use    */
          277 SGD_Sgdssi2,          /* Reserved for subsystem use    */
          281 SGD_Sgdcnfrm,         /* Old location: confirmed status*/
          289 SGD_Sgdgbkuf,         /* Guaranteed backup freq        */
          293 SGD_Sgdtblgr,         /* OAM Table Space ID            */
          3Ø1 SGD_Sgdoamfl,         /* OAM flags, see below          */
          3Ø2 SGD_Reserved3,        /* Reserved                      */
          3Ø3 SGD_Sgdcylst,         /* OAM Cycle Start time (hrs)    */
          3Ø4 SGD_Sgdcyled,         /* OAM Cycle End time (hrs)      */
          3Ø5 SGD_Sgdvolft,         /* Volume Full Threshold bit     */
          3Ø7 SGD_Sgddrvst,         /* Drive Start Threshold bit     */
          3Ø9 SGD_Sgdolibs,         /* Libraries(optical,tape)       */
          565 SGD_Sgdsysof,         /* Offset to system data         */
          569 SGD_Sgdsysln,         /* length of system data         */
          573 SGD_Sgdosysn,         /* OSMC system name              */
          581 SGD_Reserved4,        /* Reserved                      */
          593 SGD_Sgdsysdt          /* System related data           */
```

```
SGD_Sgdnmlen = c2d(SGD_Sgdnmlen)
SGD_Sgdftype = c2d(SGD_Sgdftype) /* Storage Group type */
                                 /* Ø - Pool */
                                 /* 1 - VIO */
                                 /* 2 - Dummy */
                                 /* 3 - Object */
                                 /* 4 - Object Backup */
                                 /* 5 - Tape */
                                     /* HSM auto backup, 1=YES,Ø=NO   */
SGD_Flg_Sgdfabup = bitand(SGD_Sgdflags,x'8Ø') = x'8Ø'
                                     /* Auto migration, 1=YES, Ø=NO   */
SGD_Flg_Sgdfamig = bitand(SGD_Sgdflags,x'4Ø') = x'4Ø'
                                     /* Auto dump, 1 = YES, Ø = NO    */
SGD_Flg_Sgdfadmp = bitand(SGD_Sgdflags,x'2Ø') = x'2Ø'
                                     /* Thresholds specified 1=Y,Ø=N  */
SGD_Flg_Sgdfthrs = bitand(SGD_Sgdflags,x'1Ø') = x'1Ø'
                                     /* Guaranteed backup freq 1=Y,Ø=N*/
SGD_Flg_Sgdfgbku = bitand(SGD_Sgdflags,x'Ø8') = x'Ø8'
                                     /* Guaranteed backup freq 1=NOLIM*/
SGD_Flg_Sgdgbnol = bitand(SGD_Sgdflags,x'Ø4') = x'Ø4'
                                     /* Int Mig, 1=Yes, Ø=No          */
SGD_Flg_Sgdfimig = bitand(SGD_Sgdflags,x'Ø2') = x'Ø2'
                                     /* Prim space AM 1=Yes, Ø=No     */
SGD_Flg_Sgdfpsm  = bitand(SGD_Sgdflags,x'Ø1') = x'Ø1'
                                     /* OAM Cycle start/end given     */
SGD_Flg_Sgdfcys  = bitand(SGD_Sgdoamfl,x'8Ø') = x'8Ø'
                                     /* Volume Full Threshold bit     */
SGD_Flg_Sgdfvlft = bitand(SGD_Sgdoamfl,x'4Ø') = x'4Ø'
                                     /* Drive Start Threshold bit     */
SGD_Flg_Sgdfdrst = bitand(SGD_Sgdoamfl,x'2Ø') = x'2Ø'
                                     /* Vol full write er given       */
SGD_Flg_Sgdvffer = bitand(SGD_Sgdoamfl,x'1Ø') = x'1Ø'
                                     /* Vol full Write error bit      */
SGD_Flg_Sgdvferr = bitand(SGD_Sgdoamfl,x'Ø8') = x'Ø8'
SGD_SmsStat. = ""                     /* SMS Status */
                                      /* Ø - No status given */
                                      /* 1 - Enabled */
                                      /* 2 - Quiesce/All */
                                      /* 3 - Quiesce/New */
                                      /* 4 - Disabled/All */
                                      /* 5 - Disabled/New */
SGD_SmsCStat. = ""                    /* Confirmed SMS Status */
                                      /* Same as Smsstat */
do j = 1 to 8 /* Max systems 256 */
   SGD_SmsStat.j = c2d(substr(SGD_SistSTAT,((j-1)*8)+1,1))
   SGD_SmsCStat.j = c2d(substr(SGD_SistSTAT,((j-1)*8)+2,1))
   end
return
```

Salvador Carrasco
Computer Room Coordinator (Spain)

Serena has announced enhancements to its StarTool Family Product Suite.

The company has released Version 7.3 of Serena StarTool FDM (its file and data manager), and Version 3.1 of Serena StarTool APM (its application performance manager), which are compatible with the most recent versions of z/OS, further extending the company's support for IBM's latest architecture.

Later this year, Serena will release the next version of Serena StarTool ATD, its application test debugger. It has already shipped StarTool DA 5.2 and Serena StarTool IOO 3.1.

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

* * *

Mainstar has announced release 6.101A of its Catalog RecoveryPlus, with enhancements that promise improved functionality in a number of core areas.

New Extract file functionality has been added to the EXPLORE command, the DIAGNOSE BCS-VVDS command no longer requires a journal data set, and the DIAG B-V reports have been improved.

There's a new keyword REMOVE for ALTER BCS BACK-POINTERS, for removing unused BCS back-pointers within VVCR/VVCN. A new keyword VVCR for ZAP PRINT command will initiate a print of VVCR, VVCN, and VVCM records.

A ZAP PATCH command enhancement now allows users to change a primary key value in a BCS and there's added ISPF support for BACKUP DSN, RECOVER DSN, and EXPLORE. The RECOVER DSN command now allows changes to the values for space allocation, free space, and CI size during the recovery.

Also new is BACKUP and RECOVER command support for compressed data sets and an Installation Verification Process (IVP) is supplied with the product. The Installation & Maintenance Guide shows how to use it for verification of new releases. Meanwhile, a new ISPF panel Catalog Search is for searching for specific catalog information and there's improved RECOVER BCS command reporting, now displaying totals by record type.

For further information contact:
Mainstar Software, PO Box 4132, Bellevue, WA 98009-4132, USA.
Tel: (425) 455 3589.
URL: http://www.mainstar.com.

* * *

IBM has announced its Migration Utility for z/OS and OS/390, allowing sites to create standard COBOL reports using Computer Associates' Easytrieve Plus language without Easytrieve Plus installed, and which runs in place of the Easytrieve run-time interpreter.

The applications can then be modified, maintained, and enhanced via standard COBOL programming.

For further information contact your local IBM representative.
URL: http://www.ibm.com/software/ad/migration.