196

MVS update

January 2003

In this issue

Published by

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

North American office

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

Subscriptions and back-issues

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

MVS Update on-line

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

Editor

Trevor Eddolls

Disclaimer

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

Contributions

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

Printed in England.

Introducing and backing out software changes with minimum disruption

Or Changing parameters and bringing software changes online without an IPL.

PREFACE

There are two main reasons for writing this article.

First, in my current workplace we are using a technique of introducing software changes (updating parameters) without an IPL, and wish to continue using this technique.

At the moment there is no official way to change a parameter without an IPL. The writers of the Redbook *Parallel Sysplex – Managing Software for Availability* introduced a program (SYMUPDTE) that allows this. We believe that when more IBM customers use this 'unofficial' technique there is every chance that IBM will recognize its importance and introduce it officially.

Secondly, as I am fairly new to the concepts and tools, through the research and preparation of this article, I personally will gain a better understanding of what is involved.

INTRODUCTION

The introduction of a new version of a software product involves a great amount of effort. Datasets need to be created, copied, modified, and authorized. Procedures and JCL need to be updated, tested, corrected, and further tested. When everything is ready on one LPAR, it is usually necessary to transfer everything to one or more production LPARs. When something doesn't quite work as expected (a common PROC/JCL nightmare), it is preferable to be able to reverse the process as quickly and with as minimal a distruption as possible (normally at least the changes in procedures and JCL would need reversing).

On top of all this, it is quite often necessary to perform one IPL to implement the changes and another to reverse them, should it be necessary.

At the beginning of 2002 I started work with my current employer. Within the first few weeks I received new software versions from one of our suppliers. I immediately noticed that things were very different from with my previous employer when it came to introducing these software changes.

Here it was possible to introduce software changes on one LPAR, test them fully, and, with the minimum disruption, implement the new versions on one or more other LPARs. If a problem occurred the whole process could be quickly and exactly reversed with an absolute minimum of disruption.

WORKING EXAMPLE

To describe the technique, it is easier for me to use a working example, and therefore I will describe the update of the STROBE (Compuware) product from Version 2.3.1 to 2.5.0.

Existing definitions

&RELSTR, the parameter for the level of the STROBE release defined in SYS1.PARMLIB(IEASYM00).

Dataset aliases as defined using IDCAMS:

```
DEFINE ALIAS(NAME(SYS4.STR.CLIST)
SYMBOLICRELATE(SYS4.STR&RELSTR..CLIST))
DEFINE ALIAS(NAME(SYS4.STR.ISPMLIB.MSGS)
SYMBOLICRELATE(SYS4.STR&RELSTR.. ISPMLIB.MSGS))
DEFINE ALIAS(NAME(SYS4.STR.ISPPLIB.PANELS)
SYMBOLICRELATE(SYS4.STR&RELSTR.. ISPPLIB.PANELS))
DEFINE ALIAS(NAME(SYS4.STR.ISPSLIB.SKELS)
SYMBOLICRELATE(SYS4.STR&RELSTR.. ISPSLIB.SKELS))
DEFINE ALIAS(NAME(SYS4.STR.MESSAGES.MSGS)
SYMBOLICRELATE(SYS4.STR&RELSTR.. MESSAGES.MSGS))
DEFINE ALIAS(NAME(SYS4.STR.LOADLIB)
SYMBOLICRELATE(SYS4.STR&RELSTR..LOADLIB))
```

Entries in SYS1.PARMLIB (IEASYM00) for IPL (all LPARs using Version 2.3.1):

```
SYSDEF     SYMDEF(&RELOS='21Ø')          /* OPERATING SYSTEM BASE       */
            SYMDEF(&RELSTR='231')        /* STROBE rel 231              */
SYSDEF      LPARNAME(LPØ1)               /* PRODUCTION                  */
            SYSNAME(SYØ1)
            SYMDEF(&LPARNUM='Ø1')
SYSDEF      LPARNAME(LPØ2)                  /* DEVELOPMENT              */
             SYSNAME(SYØ2)
             SYMDEF(&LPARNUM='Ø2')
 SYSDEF      LPARNAME(LPØ3)                 /* SYSTEMS TEST             */
            SYSNAME(SYØ3)
            SYMDEF(&LPARNUM='Ø3')
```

With the above configuration (after an IPL) the dataset alias SYS4.STR.LOADLIB points to the actual dataset SYS4.STR231.LOADLIB.

INSTALLATION

Stage 1

The supplied installation dataset is copied from the installation tape to dataset INST.STR250.CNTL using IEBCOPY. The previous version's installation dataset INST.STR231.CNTL is used as reference during the installation to show on-site modifications and may be deleted after the successful installation and implementation of the newer version.

Stage 2

The installation is applied to the base datasets (new) with the prefix SMPT.STR250.

After the installation, these datasets are copied to the relevant datasets with the prefix SYS4.STR250.

PTFs, when received, are applied directly to the base SMPT datasets and then copied to the next higher prefix. For example ,SYS4.STR250A, then SYS4.STR250B, then SYS4.STR250C, etc.

The datasets of the last two versions of a software product are generally kept to allow a quick backout should any problems with the new version be detected. Often obscure problems are first

detected after several weeks; this method allows for the old version to be used during the correction of the new version, or if necessary a complete backout is possible with minimum intervention.

Stage 3

To test the release between IPLs or to quickly switch back to a previous release, the program SYMUPDTE is used to change the parameter &RELSTR.

JCL:

```
//*
//*
//     INCLUDE MEMBER=$JCLSVAR
//*
//SYMBOL1 EXEC ISPFSYS,EX=SYMUPDTE,EXPARM='SY03 RELSTR=250'
```

For acceptance tests, over a long period of time (over one or more IPLs), the entries in IEASYM00 would need to be changed as follows so that the 'SYSTEMS TEST' LPAR runs Version 2.5.0 and the other LPARs Version 2.3.1.

```
SYSDEF     SYMDEF(&RELOS='210')          /* OPERATING SYSTEM BASE     */
SYSDEF      LPARNAME(LP01)               /* PRODUCTION                */
            SYSNAME(SY01)
            SYMDEF(&LPARNUM='01')
            SYMDEF(&RELSTR='231')        /* STROBE rel 231            */
SYSDEF      LPARNAME(LP02)               /* DEVELOPMENT               */
             SYSNAME(SY02)
             SYMDEF(&LPARNUM='02')
             SYMDEF(&RELSTR='231')       /* STROBE rel 231            */
 SYSDEF      LPARNAME(LP03)              /* SYSTEMS TEST              */
            SYSNAME(SY03)
            SYMDEF(&LPARNUM='03')
            SYMDEF(&RELSTR='250')        /* STROBE rel 250            */
```

Stage 3a

When SYMUPDTE is used, the relevant linklist entries must be activated (after deleting the previous and adding the new version dataset entries) using SETPROG.

The command from the console is:

```
SET PROG=SR
```

Member SYS2.MODIFY.PARMLIB(PROGSR) contains:

```
APF DELETE DSNAME(SYS4.STR231.LOADLIB) SMS
APF ADD    DSNAME(SYS2.STR25Ø.LOADLIB) SMS
LNKLST DEFINE NAME(LNKLSTSTR) COPYFROM(CURRENT)
LNKLST DELETE NAME(LNKLSTSTR) DSNAME(SYS2.STR231.LOADLIB)
LNKLST ADD    NAME(LNKLSTSTR) DSNAME(SYS2.STR25Ø.LOADLIB)
LNKLST ACTIVATE NAME(LNKLSTSTR)
```

Stage 4

Restart the subsystem from the console (S STROBE).

SUMMARY

The method described has many advantages over previous techniques:

1 It is simple to implement.

2 It minimizes interruptions to application availability.

3 There are no JCL changes on switching from one version to another.

4 All copying can be done prior to switching.

5 No datasets need to be recatalogued or renamed.

6 It provides quick and easy backout: stop the subsystem, change the parameter, activate the linklist, and restart the subsystem.

REFERENCES

IBM Redbook, *Parallel Sysplex – Managing Software for Availability (Section 3.5)*, ISBN 0738415472 IBM Form Number SG24-5451-00, www.redbooks.ibm.com.

NASPA article: *Storage Strategies* by Steve Pryor, http://www.naspa.com/PDF/2001/0901%20PDF/T0109009.pdf, www.naspa.com

Rolf Parker
Systems Programmer (Germany)

A multi-platform/multi-feature solution

A well-known problem for us COBOL programmers is how to calculate the length of a declared structure before compilation time. There's also a great number of solutions of every possible kind. I have written some REXX functions for the IBM mainframe platforms.

What I present here is probably not the ultimate solution. I offer it as an example using the strengths of a typical network configuration, namely:

- I use a PC as workstation – the operating systems is Windows NT, running an emulator (NS/Elite, in my case) to emulate a 3270 terminal.
- The host side runs OS/390.

Other components include the following:

- Under WinNT, Visual Basic (upwards from Version 5).
- Under OS/390, ISPF and REXX.
- A special component is the Workstation Agent from IBM; it's client/server software to use the functionalities of the client (WinNT – there are client components for other operating systems as well). These functions include downloading a file or starting a program on the client.
- Another special component is the freeware SIZER from Progeni (available for download from www.progeni.com). It's a small program that calculates the length of a structure with every possible attribute, like OCCURS etc.

How does it work?

- On the host side one must edit the the text, which includes the COBOL declaration under ISPF.
- The edit macro, COBLEN, must be called from the command line. If the labels .b and .e (begin-end) are given, it calculates

the length of this section, otherwise the whole text is considered to be a structure.

- The macro does the following:
  - It downloads the selected text into a file in the directory *c:\temp* on the PC.
  - It starts a small Visual Basic program, ClipBoardCopy, which copies this file into the Windows clipboard. This is necessary because SIZER uses the clipboard as input.
  - It starts the program SIZER from the directory *C:\tmp\mvs*. That's the directory where the downloaded SIZER must be installed.

The output (ie the length of the structure) will be shown in a small message box. It is somewhat customizable. I think I have seen that it can also show the offsets of the fields as well, but I've succeeded only once with this 'magic' – Progeni can perhaps help to stabilize this feature if it really exists.

What are the lessons to be learned from this solution? It helps you to learn how to use the Workstation Agent for simple procedures.

As I've already said, the WSA is a free component that is delivered with OS/390. You must install it as follows:

- Download the file SYS1.ISPF.SISPGUI to your PC under the name ISPFINST.EXE.
- Execute this program on the PC *once.*
- Create a linkage with the application WSA.EXE.

This procedure might be slightly different at your shop; contact your systems programmers!

It teaches you how to combine the different components of the host and the workstation to achieve the optimal solution. You can, for example, start Excel to process your host data as a spreadsheet, or use Word to make a grammatical correction to your text document!

A final word on the Visual Basic component. Please don't be afraid to use this language even if you are a host dinosaur like myself – it ain't vodoo magic! You must set up the simplest VB project and copy my first listing into the text object. No GUI is necessary. You must create an .exe file with the name ClipBoardCopy.exe and store it in the directory *c:\Programs* (you can of course change this in the REXX function if you prefer another directory).

CLIPBOARD COPY

```
'-------------------------------------------------'
' Function to copy a textfile to the clipboard
'        (no GUI is necessary)
'-------------------------------------------------'
Private Sub Form_Load()
  Dim strFileText As String
  ' empty the Clipboard
  Clipboard.Clear
  If FileExists("c:\temp\clipBoardCopy.txt") = False Then
    MsgBox ("File does not exist!")
  Else
    ' read the textfile into a string
    strFileText = ReadFromFile("c:\temp\clipBoardCopy.txt")
    ' copy the string into the clipboard
    Clipboard.SetText strFileText, vbCFText
  End If

End Sub
'
Public Function FileExists(FileName As String) As Boolean
    ' test if the inputfile exists
    If Dir(FileName) = "" Then
        FileExists = False
    Else
        FileExists = True
    End If
End Function
'
Public Function ReadFromFile(sFile As String) As String
    Dim fs As FileSystemObject
    Dim a As TextStream
    Set fs = CreateObject("Scripting.FileSystemObject")
    Set a = fs.OpenTextFile(sFile)
    ReadFromFile = a.ReadAll
    a.Close
```

```
End Function
```

COBLEN EDIT MACRO

```
/******************************** REXX ********************************/
/*  Ver.1.Ø                                                           */
/*  EDIT Macro COBLEN                                                 */
/*                                                                    */
/*  Description :                                                     */
/*  ------------                                                      */
/*  This edit macro is used to calculate the length of a COBOL        */
/*  structure.                                                        */
/*  If the labels .a and/or .e are set, it calculates the size of     */
/*  the structure between these two labels (both are options, if one  */
/*  fails, it will be from the first or to the left line processed).  */
/*                                                                    */
/* -It uses the Workstation Agent(IBM) for the download               */
/* -It uses the Freeware component progeni.exe from the company       */
/*               Progeni Corporation,Norcross, GA, USA                */
/*  at www.progeni.com                                                */
/*                                                                    */
/*  WARNING: when the message-box of the progeni component shown,     */
/*  please push on the 'recalculate' button if the size is not        */
/*  automatically shown...                                            */
/*                                                                    */
/*  ----------+-------------------------------------------+--------  */
/*  Parameter !               Description                 ! Default  */
/*  ----------+-------------------------------------------+--------  */
/*  parameter ! parameter-Description                     !          */
/*  ----------+-------------------------------------------+--------  */
/*  none      +                                           +          */
/*  ----------+-------------------------------------------+--------  */
/*                                                                    */
/**********************************************************************/
/*                                                                    */
/*                            Changes:                                */
/*                            --------                                */
/*                                                                    */
/**********************************************************************/
  address ISPEXEC "CONTROL ERRORS RETURN "
  address ISREDIT
  "MACRO (PARMS) NOPROCESS"
  parse var parms param
  /* ---------------------------------------------------------*/
  /*  Copy  the whole/selected edited text into the stack      */
  /* ---------------------------------------------------------*/
  "ISREDIT PROCESS"
  "ISREDIT (ENV) = USER_STATE"
  "ISREDIT (MEMBER) = MEMBER"
```

```
"ISREDIT (DATASET) = DATASET"
"ISREDIT (PROFILE) = PROFILE"
if member ¬= '' & member ¬= 'MEMBER' /* Partitioned DS */
then
  DatasetName = "'"dataset"("member")'" /* Sequential DS */
else
  DatasetName = "'"dataset"'"
Address TSO
hostdsn = "'"userid()".DOWN'"
$ = listdsi(hostdsn' DIR')
rea = sysreason
if sysreason = Ø
then
  "ALLOC F(DOWN) DA('" || userid() || ".DOWN') OLD REUS "
else
  "ALLOC F(DOWN) DA('" || userid() || ".DOWN') NEW CATALOG " || ,
  "SPACE(1Ø 5Ø) CYL" || ,
  " BLKSIZE(Ø) LRECL(2ØØØ) RECFM(V B)"
"ISREDIT (LINE1) = LINENUM .A"
if rc > Ø
then
  "ISREDIT (LINE1) = LINENUM .ZFIRST"
"ISREDIT (LINE2) = LINENUM .E"
if rc > Ø
then
  "ISREDIT (LINE2) = LINENUM .ZLAST"
text = 'DataSet >'DataSetName'< on 'Date()' at 'Time()' downloaded'
push text
"EXECIO 1 DISKW DOWN"
do i = line1 to line2
  "ISREDIT (TEXT) = LINE "i
  push text
  "EXECIO 1 DISKW DOWN"
end
"EXECIO Ø DISKW DOWN (FINIS)"
/* ---------------------------------------------------------------*/
/*  Transfer text (=copy-Structure) to the workstation            */
/* ---------------------------------------------------------------*/
parse var DatasetName "'" DatasetName "'"
DatasetName = translate(DatasetName,'_ ','()')
dir         = "C:\TEMP\clipBoardCopy.txt"
address ISPEXEC
"FILEXFER HOST(HOSTDSN) WS(DIR) TO(WS) TEXT MAKEPATH(YES)"
if rc = Ø
then do
  "FREE F(DOWN)"
end
else do
  if rc = 1Ø
  then
```

```
      say 'Please establish the Workstation-Connect'
    else
      say 'Filetransfer cancelled,RC:'rc
  end

/*-----------------------------------------------------------------*/
  /* Call the copy program to copy the downloaded textfile into the */
  /* clipboard                                                      */
/*-----------------------------------------------------------------*/
  wscmd = "C:\Programs\clipBoardCopy.EXE "
  "ISPEXEC SELECT WSCMDV(WSCMD) MODELESS"
/*-----------------------------------------------------------------*/
  /* Call the COBOL Utility program to calculate the size of the   */
  /* structure                                                      */
/*-----------------------------------------------------------------*/
  wscmd = "C:\tmp\mvs\sizer.exe "   /* Progeni executable */
  "ISPEXEC SELECT WSCMDV(WSCMD) MODELESS "
/*-----------------------------------------------------------------*/
  /* End , return with CC=Ø                                         */
/*-----------------------------------------------------------------*/
  "ISREDIT RES"
return Ø
```

Gabor Markon
Systems Engineer
HVB Systems (Germany)

Expanding MFS (IMS/DC) for EQU statements

PROBLEM

The place where I currently work uses IMS DC. We have a large number of MFS members that have EQU statements coded within each member. Also, each MFS member holds definitions for a number of screens. It is very difficult and time-consuming to read and interpret these MFS statements, especially when debugging a production problem or doing maintenance/ enhancements on the member. Just imagine a DFLD statement having these attributes: P.(03,35),L.10,PNHN,R.

SOLUTION

This EXEC will translate and replace all the referenced values with EQU values that were defined using EQU statements. You can use this EXEC either in ISPF/EDIT or in TSO. If you call this EXEC while editing the member, the expanded statements are inserted as NOTES under the original statements – perhaps you just want to see the expanded values. Typing RESET will clear these NOTES statements. If you call this EXEC in TSO, the EXEC will prompt you for the input file and the output is written to a sequential file. If you pass any non-blank character as a parameter while in Edit mode, it is the same as calling the EXEC in TSO – ie the output is written to a sequential file.

```
/*  -------- REXX Exec ------------- */
/* If this character is not | logical OR*/
/* please make a a global change to     */
/* modify this character to logical OR. */
/*  ---------------------------------- */
/*  Sometimes MFS statements have EQU   */
/*  statements and it is difficult      */
/*  to read. This macro will expand     */
/*  all FLDs referred by EQU statements */
/*  If called within EDIT, the expanded */
/*  statements are displayed as NOTES   */
/*  If called from TSO, the whole MFS   */
/*  with expanded statements is written */
/*  to a dataset.                       */
/*  ---------------------------------- */
Trace O
Address ISREDIT
'MACRO (xParams) PROCESS'
If Rc=Ø Then Call ReadFileInEdit
  Else Call ReadFileInTSO
Call InitVar
Call SegregateData
Call ReadNReplace
Call BuildAgain
If sNewFile='Y' Then Call CreateFile
 Else Call DisplayReset
Exit
ReadFileInEdit:
/*************/
sNewFile='N'
      /* sNewFile='N' --> Expand The Statements As Notes Within File */
      /* sNewFile='Y' --> Expand The Statements & Write a New File */
If xParams='' Then sNewFile='N'
```

```
  Else sNewFile='Y'
Address ISREDIT
'(nLast) = LINENUM .ZLAST'
Do nCountI=1 By 1 Until nCountI=nLast
 '(xData) = LINE (nCountI)'
 xCards.nCountI=xData
 End
Return
ReadFileInTSO:
/*************/
Address TSO
Arg xMfsCrds .
If xMfsCrds="" Then Do
  Say 'Please Key In The Dsn Where You Have The Image'
  Say 'If Dsn Is Not In Quotes,' Userid() 'Will Be Suffixed To It'
  Pull xMfsCrds
  If xMfsCrds='' Then Exit
  End
If Left(xMfsCrds,1)="'" Then Nop
 Else xMfsCrds="'"Userid()"."xMfsCrds"'"
xStrpName="'"Strip(xMfsCrds,'B',"'")"'"
xAvail=Sysdsn(xStrpName)
If xAvail<>'OK' Then Do
  Say '** Error **' xMfsCrds 'Is Not Present'
  Say 'Program Abandoned'
  Exit 9999
  End
xNazBeg = Msg("OFF")
"FREE DD(xDdMfs)"
xAskha = Msg(xNazBeg)
"ALLOCATE DD(xDdMfs) DSN("xMfsCrds") SHR REUSE"
If Rc<>Ø Then Do
  Say '*Error* Unable To Alloc' xMfsCrds 'Return Code Is ' Rc
  Say '        Program Aborted'
  Exit
  End
"EXECIO * xDdMfs (STEM xCards."
If Rc<>Ø Then Do
  Say '*Error* Unable To Read' xMfsCrds 'Return Code Is ' Rc
  Say '        Program Aborted'
  Exit
  End
"EXECIO * xDdMfs (FINIS"
xNazBeg = Msg("OFF")
"FREE DD(xDdMfs)"
xNazBeg = Msg("OFF")
nLast = xCards.Ø
sNewFile='Y'
Return
SegregateData:
```

```
/*************/
Do nCountI=1 By 1 Until nCountI=nLast
 xData=xCards.nCountI
 xData=Substr(xData,1,72)
 xOrgnl.nCountI=xData
 xOrig.nCountI=xData
 If Left(xData,1)='*' Then Iterate nCountI
 If sContinue='Y' Then Do
   nCountK=nCountK+1
   xSaveI.nCountK=nCountI
   If Substr(xData,72,1)=' ' Then sContinue='N'
   End
 xFld=Substr(xData,1Ø,4)
 If xFld='MFLD' | xFld='DFLD' Then Do
  nCountK=nCountK+1
  xSaveI.nCountK=nCountI
  If Substr(xData,72,1)<>'' Then sContinue='Y'
  End
 If Word(xData,2) <> 'EQU' Then Iterate
 Parse Var xData xDef . xValue
 xValue=Strip(xValue)
 If Left(xValue,1)<>"'" Then xValue="'"xValue"'"
 Interpret 'xVal.'xDef'='xValue
 nCountJ=nCountJ+1
 xEqu1.nCountJ=xDef'.'
 xEqu2.nCountJ=xDef' '
 End
nSaveJ=nCountJ
Return
ReadNReplace:
/***********/
Do nCountI=1 By 1 Until nCountI=nCountK
  M=xSaveI.nCountI
  xTemp=xOrig.M
  If Pos("'",xTemp)>Ø Then Do
    Do nCountJ=16 By 1 To 71
     xCh=Substr(xTemp,nCountJ,1)
     If xStComa='Y' & xCh="'" Then xStComa='N'
     If xStComa='' & xCh="'" Then xStComa='Y'
     If xStComa<>'' Then xTemp=Overlay('~',xTemp,nCountJ,1)
     If xStComa='N' Then xStComa=''
     End
    End
  If Pos("(",xTemp)>Ø Then Do
    Do nCountJ=16 By 1 To 72
     xCh=Substr(xTemp,nCountJ,1)
     If xStBrkt='Y' & xCh=")" Then xStBrkt='N'
     If xStBrkt='' & xCh="(" Then xStBrkt='Y'
     If xStBrkt<>'' Then xTemp=Overlay('~',xTemp,nCountJ,1)
     If xStBrkt='N' Then xStBrkt=''
```

```
   End
  End
Parse Var xTemp   1 xTemp1 16  xTemp2
Parse Var xOrig.M 1 xOrig1 16  xOrig2
Parse Var xTemp2 xTemp2 .
nLen=Length(xTemp2)
xOrig2=Substr(xOrig2,1,nLen)
nCountJ1=Ø
Do While Pos(',',xTemp2)>Ø
 nPos=Pos(',',xTemp2)
 nCountJ1=nCountJ1+1
 xTemp.nCountJ1=Substr(xTemp2,1,nPos-1)
 xOrig.nCountJ1=Substr(xOrig2,1,nPos-1)
 xTemp2=Substr(xTemp2,nPos+1)
 xOrig2=Substr(xOrig2,nPos+1)
 End
nCountJ1=nCountJ1+1
xTemp.nCountJ1=xTemp2
xOrig.nCountJ1=xOrig2
Do nCountJ2=1 By 1 Until nCountJ2=nCountJ1
  xTrans=xTemp.nCountJ2
  xMorig=xOrig.nCountJ2
  xRest=xMorig
  nPosDot=Pos('.',xTrans)
  If nPosDot>Ø Then Do
    xMay1=Substr(xTrans,1,nPosDot)
    Do nCountJ3=1 By 1 Until nCountJ3=nSaveJ
      xMak=xEqu1.nCountJ3
      If xMak=xMay1 Then Do
        xRest=Substr(xMorig,nPosDot+1)
        xMak=Strip(Translate(xMak,'','.'))
        xRest=xVal.xMak||xRest
        Leave nCountJ3
        End
     End
    End
   Else
    Do nCountJ3=1 By 1 Until nCountJ3=nSaveJ
      xMak=xEqu2.nCountJ3
      xMak=Strip(xMak)
      If xMak=xTrans Then Do
        xRest=xVal.xMak
        Leave nCountJ3
        End
     End
    If Length(xOrig1)<16 Then xOrig1=xOrig1||xRest
     Else xOrig1=xOrig1','xRest
    End
If Left(xOrig1,71)<>Left(xOrig.M,71) Then xRepl.M=xOrig1
End
```

```
Return
BuildAgain:
/*********/
nCountJ=Ø
Do nCountI=nLast By -1 Until nCountI=1
 If xRepl.nCountI='?' Then Do
   If sNewFile='Y' Then Do
     nCountJ=nCountJ+1
     xNewDat.nCountJ=xOrgnl.nCountI
     End
   Iterate nCountI
   End
 Parse Var xOrig.nCountI 1 xPart1 9 xRest
 If Length(xRepl.nCountI)>71 Then Do
   If Pos(',POS=',xRepl.nCountI)<71 Then Do
     Parse Var xRepl.nCountI xPart1 'POS=' xPart2
     xData=Copies(' ',14) 'POS='||xPart2
     If sNewFile='Y' Then Do
       nCountJ=nCountJ+1
       xNewDat.nCountJ=xData
       End
      Else 'Line_After' nCountI   '= NoteLine (xData)'
     xData=Left(xPart1,7Ø) '*'
     If sNewFile='Y' Then Do
       nCountJ=nCountJ+1
       xNewDat.nCountJ=xData
       End
      Else 'Line_After' nCountI   '= NoteLine (xData)'
     End
    Else Do
     Parse Var xRepl.nCountI 1 xPart1 71 xPart2
     xData=xPart1'*'
     If sNewFile='Y' Then Do
       nCountJ=nCountJ+1
       xNewDat.nCountJ=Copies(' ',14) xPart2
       nCountJ=nCountJ+1
       xNewDat.nCountJ=Copies(' ',14) xData
       End
      Else Do
       'Line_After' nCountI   '= NoteLine' Copies(' ',14) xPart2
       'Line_After' nCountI   '= NoteLine (xData)'
       End
     End
    End
  Else Do
   xData=xRepl.nCountI
   If sNewFile='Y' Then Do
     nCountJ=nCountJ+1
     xNewDat.nCountJ=xData
     End
```

```
    Else 'Line_After' nCountI   '= NoteLine (xData)'
   End
 End
Return
CreateFile:
/*********/
xDfltDisp='NEW LRECL(8Ø) SPACE(8) DSORG(PS) RECFM(F,B) TRACKS RELEASE'
Address TSO
xNazBeg = Msg("OFF")
"DELETE" xFileName
"FREE DD(MFSXPND)"
"ALLOCATE DD(MFSXPND) DSN("xFileName")" xDfltDisp
If Rc<>Ø Then Do
  Say '*Error* Unable To Alloc' xFileName 'Return Code Is ' Rc
  Say '        Program Aborted'
  Dropbuf
  Exit
  End
Do nCountI=nCountJ By -1 Until nCountI=1
 Queue xNewDat.nCountI
 If Length(xNewDat.nCountI)>8Ø Then Say nCountI xNewDat.nCountI
 End
Queue ''
'EXECIO * DISKW MFSXPND'
If Rc<>Ø Then Do
  Say '*Error* Unable To Write' xFileName 'Return Code Is ' Rc
  Say '        No FIle Created -- Program ABorted'
  Dropbuf
  Exit
  End
"EXECIO Ø DISKW MFSXPND (FINIS"
"FREE DD(MFSXPND)"
xAsk=MSG(xNazBeg)
xMsg1='+-------------------------------------------+'
nLen=Length(xMsg1)
xMsg2='|' Center('Your Expanded MFS Is In',nLen-2)
xMsg2=Overlay('|',xMsg2,nLen)
xMsg3='|' Center(xFilename,nLen-4)
xMsg3=Overlay('|',xMsg3,nLen)
Say
Say xMsg1
Say xMsg2
Say xMsg3
Say xMsg1
Return
DisplayReset:
/***********/
zedsmsg='Type RESET To Clear'
zedlmsg='Type RESET & Press Enter To Clear The Notes'
Address ISPEXEC 'SETMSG MSG(ISRZØØ1)'
```

```
Return
InitVar:
/******/
nCountJ=Ø
nCountK=Ø
sContinue='N'
xOrgnl.='?'
xOrig.='?'
xRepl.='?'
xStBrkt=''
xStComa=''
xVal.='?'
xFileName="'"Userid()".ASKNB.EXPANDED.MFS'"
Return
```

Moyeen A Khan
Systems Programmer
Decision Consultants (USA)

Simplify master catalog operations across different partitions

Most OS390/MVS systems programmers work on two or more MVS images sharing the same DASD strings at one time.

I often report problems caused by HSM activity in a shared environment, because HSM may migrate datasets catalogued in the mastercatalog (eg Serverpac installation libraries), but which are accessible by other partitions too! So, I've got inconsistencies with the mastercatalog entries, because CATALOG.MVSA.CAT001 states that dataset is MIGRAT and CATALOG.MVSB.CAT002 says that the same one is on ML0 VOLUME!

Another kind of problem is when I duplicate a SYS1 or another 'mastercatalogued' dataset; I catalog it in the mastercatalog, but it is reachable by other images only via UNIT and VOLSER, unless I catalog it in the other mastercatalogs.

So, I must catalog or uncatalog the dataset in the mastercatalog of another image. It seems a simple operation, but… do I remember the command DELETE NOSCRATCH? Yes, I remember the name of the SYSID of the test or production MVS image, but what about the MASTERCATALOG name? Do I remember the dataset device type (UNIT)? And anyway, isn't it annoying to have to type a long command such DEFINE NONVSAM etc etc etc?

A SIMPLE SOLUTION USING FIVE CLISTS

Assume we have an IBM 9672 computer with four IPLable PR/SM MVS images – MVSA (production), MVSB (test), MVSC (development), and MVSD (for systems programmers' use, implementation, new installations). All MVS images share the same DASD strings. We need to catalog a library called SYS1.LINKLIB.NEWCOPY without an alias in all four ICF mastercatalogs. I am logged on MVSA TSO/ISPF.

These are the names of the mastercatalogs:

- Sysid MVSA – CATALOG.MVSA.CAT001
- Sysid MVSB – CATALOG.MVSB.CAT002
- Sysid MVSC – CATALOG.MVSC.CAT003
- Sysid MVSD – CATALOG.MVSD.CAT004

It is important that you modify the CLISTs to reflect your installation naming, especially the ????? lines (see the CLIST code below).

First of all, I must connect all the other partition's mastercatalogs to my partition mastercatalog, so it can see all of them as usercatalogs. The IMPCONN CLIST will IMPORT CONNECT in the mastercatalog of any usercatalog.

Use my IMPCONN CLIST (only in 'Dataset List Utility' ISPF 3.4 panel ISRUDSL0, left of the catalog name) by:

```
%IMPCONN <catalog>
```

I am logged on to MVSA TSO/ISPF, so I must connect the other

three as usercatalogs:

```
%IMPCONN CATALOG.MVSB.CATØØ2
%IMPCONN CATALOG.MVSC.CATØØ3
%IMPCONN CATALOG.MVSD.CATØØ4
```

To reverse the operation, use the EXPDISC CLIST to do EXPORT DISCONNECT of any usercatalog from the mastercatalog.

Use my EXPDISC CLIST (use it in 3.4 or elsewhere) by:

```
%EXPDISC <catalog>
```

For example:

```
%EXPDISC CATALOG.MVSB.CATØØ2
```

Then, use my CATAL CLIST to catalog a non-VSAM dataset (only in 'Dataset List Utility' ISPF 3.4 panel ISRUDSL0, left of the dataset name) by:

```
%CATAL <nonvsam-dataset> <system-id>
```

Go to the 3.4 panel, choose Dsname Level . . . SYS1.LINKLIB .NEWCOPY and *Enter*. Now, to the left of the to dataset name, enter:

```
%CATAL / MVSB
```

then repeat it for MVSC and MVSD. Simple, isn't it? Look at the example:

```
Command - Enter "/" to select action              Message       Volume
          SYS2.LINKLIB                                          SHRØ12
 %CATAL / MVSB LINKLIB.NEWCOPY                                  SHRØ12
```

At a successful end, dataset SYS1.LINKLIB.NEWCOPY will be catalogued in MVSB mastercatalog.

You can use the UNCAT CLIST to uncatalog a non-VSAM dataset from another MVS mastercatalog (use it in 3.4 or elsewhere):

```
%UNCAT <nonvsam-dataset> <system-id>
```

For example:

```
%UNCAT 'SYS1.LINKLIB.NEWCOPY' MVSC
```

At a successful end, dataset SYS1.LINKLIB.NEWCOPY will be uncatalogued from the MVSB mastercatalog.

Last but not least: do you want to list other mastercatalog entries, just using the partition system-id? To do so, use my LST CLIST:

```
%LST <nonvsam-dataset> <system-id>
```

For example:

```
%LST 'SYS1.LINKLIB.NEWCOPY' MVSD DEBUG
```

You can list all CLIST instructions to locate and remove errors – just add the DEBUG parameter on every CLIST.

LST CLIST

```
PROC 2 &DSN &ID DEBUG
/*- SETUP FOR DEBUG IF REQUESTED ----------------------------------*/
  CONTROL   MSG NOLIST NOFLUSH END(ENDO) NOCONLIST NOPROMPT
  IF &DEBUG  =  DEBUG THEN +
   CONTROL MSG LIST NOFLUSH END(ENDO) PROMPT SYMLIST CONLIST
/*- END OF SETUP ---------------------------------------------------*/
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/*                                                                 */
/*   LST:    TO LIST ENTRIES FROM ANOTHER MASTER CATALOG.          */
/* WARNING:  PREPARE THE CLIST WITH YOUR OWN CATALOG NAMES!        */
/*           SUPPLY ID WITH A SYSID RELATED TO A MASTERCATALOG.    */
/*           EG TO LIST ENTRY 'SYS1.TEST.DSN' FROM THE             */
/*               MASTERCATALOG OF THE SYSTEM 'MVSB' ENTER:         */
/*                                                                 */
/*           %LST 'SYS1.TEST.DSN' MVSB                             */
/*                                                                 */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
      SET &LL = &LENGTH(&DSN)
   SET &APICE = &STR(&SUBSTR(1,&DSN)
   IF &APICE = &STR(') THEN +
      SET &DSN = &STR(&SUBSTR(2:&LL-1,&DSN)
   ELSE IF &SYSPREF NE    THEN +
      SET &DATASET = &SYSPREF..&DATASET
           SELECT &ID
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/*                                                                 */
/*  NOW SUPPLY YOUR OWN CATALOG NAMES AND SYSTEM IDENTIFIERS       */
/*                                                                 */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
            WHEN (???A)   SET &MCAT=CATALOG.MVS.V?????A
            WHEN (???B)   SET &MCAT=CATALOG.MVS.V?????B
            WHEN (???C)   SET &MCAT=CATALOG.MVS.V?????C
```

```
             WHEN (???D)   SET &MCAT=CATALOG.MVS.V?????D
              OTHERWISE     DO
              WRITE SYSID NOT FOUND!
              EXIT CODE (4)
                          ENDO
            ENDO
WRITE SEARCHING ENTRY '&DSN' IN '&MCAT' ...
LISTC ENT('&DSN') ALL CAT('&MCAT')
SET &RC=&LASTCC
 IF &RC > Ø THEN WRITE LISTCAT '&DSN' FAILED FOR '&MCAT'
            EXIT CODE (&RC)
```

IMPCONN CLIST

```
PROC 1 USERCAT DEBUG
/*- SETUP FOR DEBUG IF REQUESTED ----------------------------------*/
  CONTROL   MSG NOLIST NOFLUSH END(ENDO) NOCONLIST NOPROMPT
  IF &DEBUG  =  DEBUG THEN +
   CONTROL MSG LIST NOFLUSH END(ENDO) PROMPT SYMLIST CONLIST
/*- END OF SETUP ---------------------------------------------------*/
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/*                                                                 */
/* IMPCONN : TO DO IMPORT CONNECT OF AN ICF USERCATALOG            */
/* WARNING!  VERSION FOR USE IN ISPF 3.4 ONLY                      */
/*           EXIT CODE  Ø = CATALOG CONNECTED TO MASTERCATALOG     */
/*           EXIT CODE 12 = CATALOG NOT CONNECTED TO MASTERCATALOG */
/*                                                                 */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
      SET &LL = &LENGTH(&USERCAT)
 SET &APICE = &STR(&SUBSTR(1,&USERCAT)
  IF &APICE = &STR(') THEN DO
      SET &USERCAT = &STR(&SUBSTR(2:&LL-1,&USERCAT)
                       ENDO
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/*                                                                 */
/* OBTAIN SYSID, DEVICETYPE AND VOLUME OF THE CATALOG TO BE IMPORTED */
/*                                                                 */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
  ISPEXEC VGET (ZSYSID ZDLDEV ZDLVOL)
  IF &ZDLDEV =  OR &ZDLVOL =   THEN DO
  WRITE *** YOU CAN USE 'IMPCONN' ONLY IN 3.4 ISPF DSLIST ***
  EXIT CODE(8)
                                  ENDO
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/*                                                                 */
/*  NOW SUPPLY YOUR OWN CATALOG NAMES AND SYSTEM IDENTIFIERS       */
/*                                                                 */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
          SELECT &ZSYSID
```

```
              WHEN (???A)   SET &MCAT=CATALOG.MVS.V?????A
              WHEN (???B)   SET &MCAT=CATALOG.MVS.V?????B
              WHEN (???C)   SET &MCAT=CATALOG.MVS.V?????C
              WHEN (???D)   SET &MCAT=CATALOG.MVS.V?????D
              OTHERWISE     DO
              WRITE SYSID NOT FOUND!
              EXIT CODE (4)
                        ENDO
              ENDO
IF &DEBUG NE DEBUG THEN SET &SYSLIST=ON
IMPORT CONNECT OBJ(('&USERCAT' VOLUME(&ZDLVOL) DEVICETYPE(&ZDLDEV))) -
       CATALOG('&MCAT')
SET &RC = &LASTCC
IF &DEBUG NE DEBUG THEN SET &SYSLIST=OFF
EXIT CODE(&RC)
```

EXPDISC CLIST

```
PROC 1 USERCAT DEBUG
/*- SETUP FOR DEBUG IF REQUESTED ----------------------------------*/
  CONTROL   MSG NOLIST NOFLUSH END(ENDO) NOCONLIST NOPROMPT
  IF &DEBUG  =  DEBUG THEN +
   CONTROL MSG LIST NOFLUSH END(ENDO) PROMPT SYMLIST CONLIST
/*- END OF SETUP ---------------------------------------------------*/
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/*                                                                 */
/* EXPDISC : TO DO EXPORT DISCONNECT OF AN ICF USERCATALOG         */
/*           EXIT CODE  Ø = CATALOG SUCCESFULLY DISCONNECTED       */
/*           EXIT CODE 12 = CATALOG NOT DISCONNECTED               */
/*                                                                 */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
      SET &LL = &LENGTH(&USERCAT)
 SET &APICE = &STR(&SUBSTR(1,&USERCAT)
  IF &APICE = &STR(') THEN DO
      SET &USERCAT = &STR(&SUBSTR(2:&LL-1,&USERCAT)
                       ENDO
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/*                                                                 */
/* OBTAIN SYSID                                                    */
/*                                                                 */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
  ISPEXEC VGET (ZSYSID)
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/*                                                                 */
/*  NOW SUPPLY YOUR OWN CATALOG NAMES AND SYSTEM IDENTIFIERS       */
/*                                                                 */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
       SELECT &ZSYSID
```

```
              WHEN (???A)   SET &MCAT=CATALOG.MVS.V?????A
              WHEN (???B)   SET &MCAT=CATALOG.MVS.V?????B
              WHEN (???C)   SET &MCAT=CATALOG.MVS.V?????C
              WHEN (???D)   SET &MCAT=CATALOG.MVS.V?????D
              OTHERWISE     DO
              WRITE SYSID NOT FOUND!
              EXIT CODE (4)
                        ENDO
            ENDO
IF &DEBUG NE DEBUG THEN SET &SYSLIST=ON
EXPORT '&USERCAT' DISCONNECT CAT('&MCAT')
SET &RC=&LASTCC
IF &DEBUG NE DEBUG THEN SET &SYSLIST=OFF
EXIT CODE(&RC)
```

CATAL CLIST

```
PROC 2 &DSN &ID DEBUG
/*- SETUP FOR DEBUG IF REQUESTED ----------------------------------*/
  CONTROL   MSG NOLIST NOFLUSH END(ENDO) NOCONLIST NOPROMPT
  IF &DEBUG  =  DEBUG THEN +
   CONTROL MSG LIST NOFLUSH END(ENDO) PROMPT SYMLIST CONLIST
/*- END OF SETUP ---------------------------------------------------*/
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/*                                                                 */
/* WARNING!  VERSION FOR USE IN ISPF 3.4 ONLY                      */
/* CATAL:    TO CATALOG A NONVSAM ENTRY IN A MASTER CATALOG.       */
/* WARNING:  PREPARE THE CLIST WITH YOUR OWN CATALOG NAMES!        */
/*           USING ID(NONE): NO ACTION IS DONE                     */
/*           USING ID(OTHER-VALUE) MEANS THAT WE WANT TO CATALOG   */
/*           THE NONVSAM ENTRY INTO ANOTHER CATALOG, EG TO DO IT   */
/*           FOR MASTERCATALOG OF THE SYSTEM 'MVSB' ENTER:         */
/*                                                                 */
/*          %CATAL / MVSB                                          */
/*                                                                 */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
  ISPEXEC VGET (ZDLDEV ZDLVOL)
            SELECT &ID
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/*                                                                 */
/*  NOW SUPPLY YOUR OWN CATALOG NAMES AND SYSTEM IDENTIFIERS       */
/*                                                                 */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
              WHEN (???A)   SET &MCAT=CATALOG.MVS.V?????A
              WHEN (???B)   SET &MCAT=CATALOG.MVS.V?????B
              WHEN (???C)   SET &MCAT=CATALOG.MVS.V?????C
              WHEN (???D)   SET &MCAT=CATALOG.MVS.V?????D
              OTHERWISE     DO
              WRITE SYSID NOT FOUND!
```

```
             EXIT CODE (4)
                        ENDO
             ENDO
 DEFINE NVSAM(NAME(&DSN) DEVT(&ZDLDEV) VOL(&ZDLVOL)) +
       CAT('&MCAT')
SET &RC=&LASTCC
   IF &RC=Ø THEN WRITE &DSN CATALOGED ON '&MCAT'
   ELSE          WRITE &DSN NOT CATALOGED ON '&MCAT'
            EXIT CODE (&RC)
```

UNCAT CLIST

```
PROC 2 &DSN &ID DEBUG
/*- SETUP FOR DEBUG IF REQUESTED ----------------------------------*/
  CONTROL   MSG NOLIST NOFLUSH END(ENDO) NOCONLIST NOPROMPT
  IF &DEBUG  =  DEBUG THEN +
   CONTROL MSG LIST NOFLUSH END(ENDO) PROMPT SYMLIST CONLIST
/*- END OF SETUP ---------------------------------------------------*/
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/*                                                                 */
/* UNCAT:    TO UNCATALOG A NONVSAM ENTRY IN A MASTER CATALOG.     */
/* WARNING:  PREPARE THE CLIST WITH YOUR OWN CATALOG NAMES!        */
/*           USE CAREFULLY!!! OTHER SYSTEMS MAY BE AFFECTED...     */
/*           SUPPLY ID WITH A SYSID RELATED TO A MASTERCATALOG.    */
/*           EG TO UNCATALOG 'SYS1.TEST.DSN' FROM THE              */
/*               MASTERCATALOG OF THE SYSTEM 'MVSB' ENTER:         */
/*                                                                 */
/*          %UNCAT 'SYS1.TEST.DSN' MVSB                            */
/*                                                                 */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
      SET &LL = &LENGTH(&DSN)
  SET &APICE = &STR(&SUBSTR(1,&DSN)
  IF &APICE = &STR(') THEN +
      SET &DSN = &STR(&SUBSTR(2:&LL-1,&DSN)
  ELSE IF &SYSPREF NE    THEN +
      SET &DATASET = &SYSPREF..&DATASET
            SELECT &ID
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
/*                                                                 */
/*  NOW SUPPLY YOUR OWN CATALOG NAMES AND SYSTEM IDENTIFIERS       */
/*                                                                 */
/* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . */
             WHEN (???A)   SET &MCAT=CATALOG.MVS.V?????A
             WHEN (???B)   SET &MCAT=CATALOG.MVS.V?????B
             WHEN (???C)   SET &MCAT=CATALOG.MVS.V?????C
             WHEN (???D)   SET &MCAT=CATALOG.MVS.V?????D
              OTHERWISE     DO
              WRITE SYSID NOT FOUND!
              EXIT CODE (4)
```

```
                        ENDO
             ENDO
DELETE '&DSN' NOSCRATCH CAT('&MCAT')
SET &RC=&LASTCC
 IF &RC=Ø THEN WRITE '&DSN' UNCATALOGED FROM '&MCAT'
 ELSE          WRITE '&DSN' NOT UNCATALOGED FROM '&MCAT'
             EXIT CODE (&RC)
```

Alberto Mungai
Senior Systems Programmer (Italy)

z/Architecture overview – part 2

This month we conclude our look at z/Architecture.

STORE SYSTEM INFORMATION

The Store System Information Instruction (STSI), which was added to the ESA/390 instruction set, will, depending on a function code that is placed in general register 0, return either of the following types of information:

- An identification of the level of the configuration that is executing the program (this is placed in general register 0).
- Information about a component or components of a configuration is stored in a system-information block (SYSIB). Additional information about the requested component or components can be specified further by specifying additional information in general register 0 and general register 1.

The STSI is illustrated in Figure 1.

The function codes are:

- 0 Current configuration level number is placed in bit positions 32-35 of general register 0 as follows:

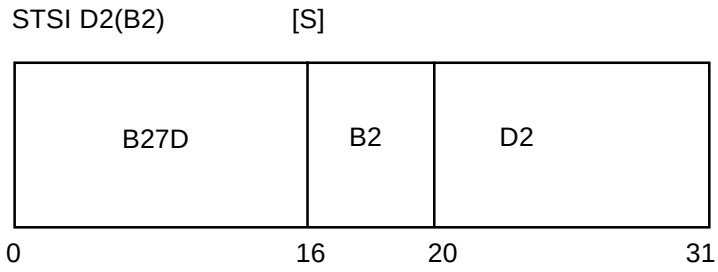


Figure 9: The Store System Information Instruction

  - 1= Basic machine
  - 2= Logical Partition
  - 3= Virtual Machine.
- 1 – Information about level 1 (the basic machine).
- 2 – Information about level 2 (a logical partition).
- 3 – Information about level 3 (a virtual machine).

The function code that determines the operation is an unsigned binary integer in bit positions 32-35 of general register 0. When the function code is non-zero (1, 2, 3) information may also be stored in a SYSIB beginning at the second-operand address. The SYSIB is 4KB in length and must begin on a 4KB boundary.

When the function code is non-zero (1, 2, 3), general registers 0 and 1 can contain additional information as follows:

- Bit positions 56-63 of general register 0 can contain an unsigned binary integer called *selector 1*, which specifies a component or components of the specified configuration as follows:
  - 1 = Information about the specified configuration.
  - 2 = Information about one or more CPUs in the specified configuration level.

- Bit positions 48-63 general register 1 can contain an unsigned binary integer called *selector 2*, which specifies the type of information requested as follows:

  When selector 1 is 1, selector 2 can have the following value:

  – 1 = Information about the specified configuration level.

  When selector 1 is 2, selector 2 can have the following values:

  – 1 = Information about the CPU executing the program in the specified configuration level.

  – 2 = Information about all CPUs in the specified configuration level.

General Register 0 format

General Register 0 format is shown in Figure 2.

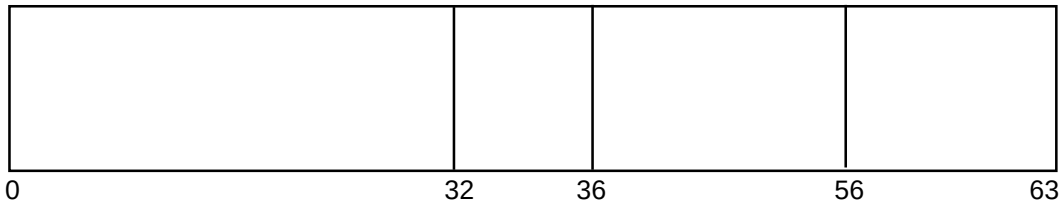


Figure 2: General Register 0 format

General Register 1 format

General Register 1 format is shown in Figure 3.

For example, when the function code = 2, selector 1 = 2 and selector 2 = 2, the SYSIB will describe the logical Partition CPUs as shown in Figure 4.

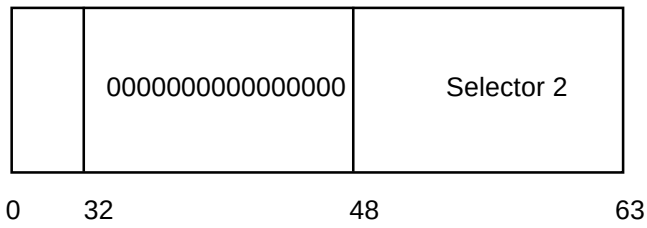


Figure 3: General Register 1 format

SET ADDRESS MODE INSTRUCTION

z/Architecture has introduced three new Set Address Mode (SAM) instructions that allow you to change the addressing mode without branching. These instructions are:

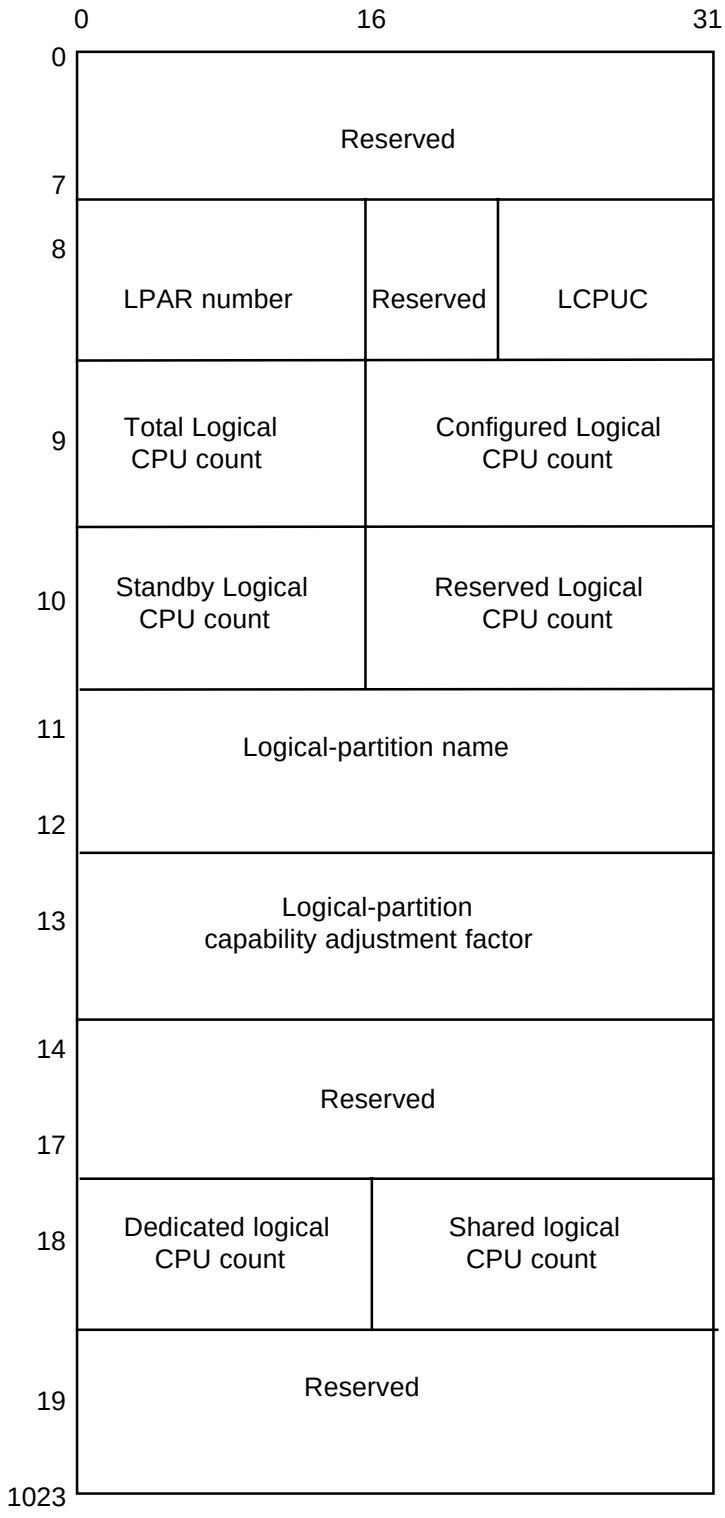
- SAM24 – set Addressing mode to 24-bit (PSW bit 31=0 PSW bit 32=0).
- SAM31 – set Addressing mode to 31-bit (PSW bit 31=0 PSW bit 32=1).
- SAM64 – set Addressing mode to 64-bit (PSW bit 31=1 PSW bit 32=1).

TEST ADDRESS MODE INSTRUCTION

The Test Address Mode (TAM) instruction sets the condition code based on the addressing mode (AMODE) currently set in the PSW as follows:

- CC= 0 – 24-bit mode
- CC= 1 – 31-bit mode
- CC= 3 – 64-bit mode.

Note: the TAM instruction has been added to the ESA/390 Instruction set.



LCPUC= Logical CPU Characteristics (dedicated, shared, utilization limit)

Figure 4: SYSIB will describe the logical partition CPUs

BRANCH AND SET MODE INSTRUCTIONS

The BASSM and BSM instruction will set 64-bit mode if the low order bit of the target address is 1. Bit 63 is left set so that the mode switch can be detected, but it is ignored in branch address generation (target addresses must always be even).

The BASSM and BSM instruction will set 24-bit or 31-bit mode based on bit 31 being 0 or 1, as in ESA/390 Architecture:

```
BASSM    R1,R2
BSM      R1,R2
```

AMODE setting bits in R2 are 32 and 63. If they are both 0, switch to 24-bit mode. If 32 is 1 and 63 is 0, switch to 31-bit mode. If 32 is *x* and 63 is 1, switch to 64-bit mode.

64-bit address generation

Sixty-four-bit addresses are generated using the value contained in a 64-bit base register, optionally taking the value contained in a 64-bit index register plus the value of the 12-bit displacement contained in the instruction. The result of this sum is then truncated on the left depending on the current addressing mode as follows:

- In 24-bit mode, the leftmost 40 bits are set to zeros.
- In 31-bit mode, the leftmost 33 bits are set to zeros.
- In 64-bit mode, the 64-bit address is not truncated.

RX Type Address Generation instruction

The RX Type Address Generation instruction comprises a base register (0 to 63), plus an Index Register (0 to 63), plus displacement (0 to 11).

Z/ARCHITECTURE INSTRUCTION SET

Many new instructions have been introduced with z/Architecture. All the existing instructions in the ESA/390 can be used in z/

Architecture mode and there are over 160 new Z/Architecture instructions. Some of the new instructions mix 64-bit and existing data types. I will not cover all the new Z/Architecture instruction set in this article; please refer to the *z/Architecture Principles of Operation* for additional information.

Mnemonics for 64-bit instructions/operations end in or contain a G, eg:

```
AGR   R4,R1Ø          Adds the 64-bit operand in R1Ø to R4.
```

Mnemonics that mix 32-bit and 64-bit operands to give a 64-bit result have GF or GH, and in a few cases GT or GC. Where:

- F = fullword
- H = halfword
- T = 31 bits
- C = character.

```
AGFR   R4,R6
```

adds the signed 32-bit operand in R6 to the 64-bit operand in R4.

```
AGHI   R4,4ØØØ
```

adds the 16-bit signed operand to the 64-bit operand in R4.

```
LGH    R4,HWORD
```

loads the signed 16-bit operand at HWORD. 64-bit result in R4.

```
LLGT   R4,F31BITS
```

loads the 31-bit operand at F31BITS. 64-bit result in R4.

There are five new instruction formats that have been introduced with z/Architecture. These are shown in Figure 5.

MODAL AND NON-MODAL INSTRUCTIONS

Two new terms have been introduced with z/Architecture that describe instruction behaviour; they are Modal and Non-Modal. The term Modal applies to instructions in which the operation is

different in 64-bit addressing mode from the operation in 24-bit or 31-bit mode. The AMODE determines the width of the output register operands. For example LA instruction operates as follows:

- In the 24-bit addressing mode, the address is placed in bit positions 40-63, bits 32-39 are set to zeros, and bits 0-31 remain unchanged.
- In the 31-bit addressing mode, the address is placed in bit positions 33-63, bit 32 is set to zero, and bits 0-31 remain unchanged
- In 64-bit addressing mode, the address is placed in bit positions 0-63.

Other examples of Modal instructions are:

- Branch and Link (BAL, BALR)
- Branch and Save (BAS, BASR)
- Branch and Save and Set Mode (BASSM)
- Branch Relative and Save (BRAS)
- Branch and Save (BSM)
- Branch Relative and Save Long (BRASL)
- Compare Logical Long (CLCL)
- Compare Logical Long Extended (CLCLE)
- Compare Logical String (CLST)
- Compare Until Substring Equal (CUSE)
- Load Address Extended (LAE)
- Load Address Relative Long (LARL)
- Move Long (MVCL)
- Move Long Extended (MVCLE)
- Move String (MVST)

RS

Opcode	R1	M3	B2	D2

The M3 field is new

RSE

Opcode	R1	R3	B2	D2		Opcode

RSE

Opcode	R1	M3	B2	D2		Opcode

RIE

Opcode	R1	M3		I2		Opcode

RIL

Opcode	R1	M3		I2

The RIL format supports signed 32-bit immediate operand

Figure 5: New instruction formats

- Search String (SRST)
- Store Pair to QUADWORD (STPG)
- Translate Extended (TRE)

- Translate And Test (TRT).

The term Non-Modal applies to instructions that perform the same operation regardless of address mode. The AMODE is only used during storage operand address generation. If the current AMODE is 64, the secondary operand can reside anywhere in the address space. If the AMODE is 31, the secondary operand must reside below the 2-gigabyte line. If the AMODE is 24, the secondary operand must reside below the 16-megabyte line. Bits 0-31 of the 64-bit registers are unexamined and unmodified. For example, the LOAD instruction (L) always loads 32 bits into bits 32-63 of the first operand. The ADD (A) instruction takes the value of a fullword in storage and arithmetically adds it to the contents of the low-order 32- bits of a general purpose register. The high order 32-bits of the general purpose register are neither interrogated nor modified. This is known as a Non-Modal 32-bit instruction.

Other examples of Non-Modal 32-bit instructions are:

- Add Register (AR)
- Add Logical Register (ALR)
- Add Logical (AL)
- Add Logical with carry (ALC)
- Load Register (LR)
- Load Reversed (LRV, LRVR)
- Multiply Logical (ML, MLR)
- Divide Logical (DL, DLR)
- Subtract Logical with borrow (SLB)
- Store Reversed (STRV)
- Rotate Left Single Logical (RLL).

The AG instruction takes the value of an 8-byte storage field and arithmetically adds it to the contents of the full 64 bits of a general purpose register. All 64 bits of the first operand are modified. This

is known as a Non-Modal 64-bit instruction.

Other examples of Non-Modal 64-bit instructions are:

- Add Register (AGR)
- Add Logical (ALG)
- Add Logical Register (ALGR)
- Add Logical with carry (ALCG)
- Divide Logical (DLG)
- Divide Logical Register (DLGR)
- Load (LG)
- Load Register (LGR)
- Load Reversed (LRVG)
- Load Reversed Register (LRVGR)
- Multiply Logical (MLG)
- Multiply Logical Register (MLGR)
- Rotate Left Single Logical (RLLG)
- Subtract Logical with Borrow (SLBG)
- Subtract Logical Register with Borrow (SLBGR)
- Store Reversed (STRVG).

The AGF instruction operates on 32-bit second operands into 64-bit first operands. The instruction takes the value of a fullword in storage and propagates the sign to extend it to 64 bits, and then arithmetically adds the sign extended value to the contents of the 64-bit general purpose register. The 32-bit second operand is internally extended to the left before the operation is performed. All 64 bits of the first operand register participate. This is known as a Non-Modal 64/32-bit instruction.

Other examples of Non-Modal 64/32-bit instructions are:

- Add (AGF, AGFR)
- Add Logical (ALGF, ALGFR)
- Compare (CGF, CGFR)
- Load (LGF, LGFR)
- Load Logical (LLGF, LLGFR)
- Load Logical Thirty One Bits (LLGT, LLGTR)
- Load Positive (LPGFR)
- Load Negative (LNGFR)
- Multiply Single (MSGF, MSGFR)
- Subtract (SGF, SGFR)
- Subtract Logical (SLGF, SLGFR).

LOGICAL IMMEDIATE INSTRUCTIONS

The new Logical Immediate Instructions load an immediate 16-bit value into any of the four halfwords of a 64-bit register. The new instructions are as follows:

- IIHH, IIHL, IILH, IILL – Insert Immediate. Insert an immediate 16-bit value into any of the four halfwords of a 64-bit register. The three other halfwords are left unchanged.
- LLIHH, LLIHL, KKILH, LLILL – Load Logical Immediate. Load an immediate 16-bit value into any of the four halfwords of a 64-bit register. The three other halfwords are cleared to zeros.
- NIHH, NIHL, NILH, NILL – AND-Immediate. Logically AND an immediate 16-bit value into any of the four halfwords of a 64-bit register. The three other halfwords are left unchanged.
- OIHH, OIHL, OILH, OILL – OR-Immediate. Logically OR an immediate 16-bit value into any of the four halfwords of a 64-bit register. The three other halfwords are left unchanged.

- TMHH, TMHL, TMLH, TMLL – Test Under Mask High/Low. Test the 16 bits of any of the four halfwords of a 64-bit register against a 16-bit immediate instruction. Note: the TMLH and TMLL instructions exist in the ESA/390 instruction set as TMH and TML.

The last two letters of the instruction mnemonic indicate which part of the Register will be acted upon:

- HH – High half's High half (bits 0-15)
- HL – High half's Low half (bits 16-31)
- LH – Low half's High half (bits 32-47)
- LL – Low half's Low half (bits 48-63).

This is illustrated in Figure 6.

LOAD LOGICAL INSTRUCTIONS

The following Load Logical instructions have been introduced in support of 64-bit:

- LLGT LLGTR – the Load Logical Thirty One instruction takes the low-order 31 bits of the 32-bit second operand and places them in bits 33-63 of the first operand register, and sets bits 0-32 of the first operand register to 33 binary zeroes. This instruction can be used to produce a clean 64-bit address from a 31-bit address regardless of the addressing mode.
- LLGF, LLGFR – Load Logical Word instruction. The 32 bits, or low-order 32 bits (LLGFR), of the second operand are loaded into the low-order 32 bits of the first operand. The high-order 32 bits of the first operand register are set to zeros.
- LLGH – Load Logical Halfword. The 16 bits of the second operand are loaded into the low-order 16 bits of the first operand register. The high-order 48 bits of the first operand register are set to zeroes.

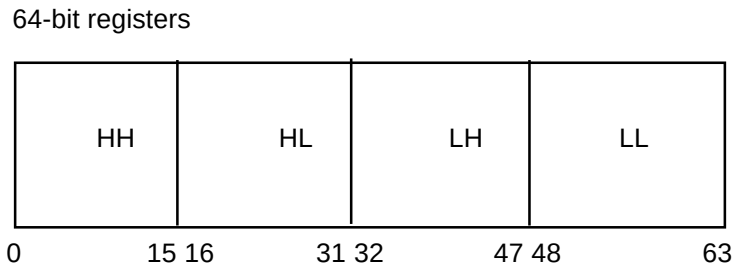


Figure 6: Logical Immediate Instructions

- LLGC – Load Logical Character. The 8 bits of the second operand are loaded into the low-order 8 bits of the first operand. The high-order 56 bits of the first operand register are set to zeroes.
- LMH – Load Multiple High. The Load Multiple High instruction works like the LM instruction except that it loads into the high-order 32 bits of the 64-bit register.

ARITHMETIC INSTRUCTIONS

The following Arithmetic instructions have been introduced in support of 64-bit:

- DSG,DSGR – Divide Single. The dividend, divisor, quotient and remainder are treated as 64-bit signed binary integers (64/64).
- DSGF, DSGFR – Divide Single. The dividend is 64-bits, the divisor is 32-bits (64/32). The quotient and remainder are treated as 64-bit signed binary integers.
- DL, DLR – Divide Logical. The dividend is 64 bits, the divisor is 32 bits (64/32); both are unsigned binary integers. The quotient and remainder are 32-bit unsigned binary integers. These two instructions are also available in the ESA/390 instruction set.

- DLG, DLGR – Divide Logical. The dividend is 128 bits, the divisor is 64 bits (128/64); both are unsigned binary integers. The quotient and remainder are 64-bit unsigned binary integers.
- ML, MLR – Multiply Logical. The multiplicand and the multiplier are 32-bit unsigned binary integers. The product is a 64-bit unsigned binary integer. These two instructions have also been made available in the ESA/390 instruction set.
- MLG MLGR – Multiply Logical. The multiplicand and the multiplier are 64-bit unsigned binary integers. The product is a 128-bit unsigned binary integer.
- ALC, ALCR – Add logical with carry. The operands, the carry, and the sum are treated as 32-bit unsigned binary integers. These two instructions have also been made available in the ESA/390 instruction set.
- ALCG, ALCGR – Add logical with carry. The operands, the carry, and the sum are treated as 64-bit unsigned binary integers.
- AG, AGR – Add. The operands are treated as 64-bit signed integers.
- AGF, AGFR – Add. The second operand is treated as a 32-bit signed binary integer and the first operand and the difference are treated as 64-bit signed binary integers.
- SLB, SLBR – Subtract Logical with borrow. The operands, borrow, and the difference are treated as 32-bit unsigned binary integers. These two instructions have also been made available in the ESA/390 instruction set.
- SLBG, SLBGR – Subtract Logical with borrow. The operands, borrow, and the difference are treated as 64-bit unsigned binary integers.
- SG, SGR – Subtract. The operands are treated as 64-bit signed integers.

- SGF, SGFR – Subtract. The second operand is treated as a 32-bit signed binary integer and the first operand and the difference are treated as 64-bit signed binary integers.

QUADWORD INSTRUCTIONS

The Load Pair From Quadword (LPQ) instruction loads a quadword second operand into an even-odd pair of general registers. The left doubleword of the quadword is loaded into R1, and the right doubleword is loaded into general register R1 + 1. The second operand must be aligned on a quadword boundary,

The Store Pair to Quadword (STPQ) instruction stores a quadword first operand at the second operand location. The first operand designates an even-odd pair of registers. The second operand must be aligned on a quadword boundary.

TEST DECIMAL

The test decimal instruction (TP) tests for valid decimal digits and valid sign codes. The instruction format is RSL:

```
TP    D1(L1,B1)
```

The results of the test are reported in the condition code as follows:

- CC=0 – all digits and sign are valid
- CC=1 – sign invalid
- CC=2 – at least one digit is invalid
- CC=3 – sign invalid and at least one digit invalid.

ADDITIONAL 64-BIT INSTRUCTIONS

The following instructions are an assortment of new instructions that have been added to the Z/Architecture instruction set:

- ICMH – Insert Characters Under Mask High. Operates on the four high-order bytes of the first operand.

- STCMH – Store Characters Under Mask High. Operates on the four high-order bytes of the first operand.
- STMH – Store Multiple High.
- LRVR, LRVRG, LRVH, LRV, LRVG – Load Reversed.
- LMD – Load Multiple Disjoint. This instruction can be used when the registers could not be stored in a single contiguous area, as on a call from an old 31-bit program to a new 64-bit program.
- LRVH, LRV, LRVG, LRVR, LRVGR – Load Reversed. The low order 2, 4 or 8 bytes of a register are loaded in byte-wise reverse order. This instruction can be used to convert two, four, or eight bytes from a 'little-endian' format to a 'big-endian' format or *vice versa*.
- STRVH,STRV, STRVG – Store Reversed. The low order 2, 4, or 8 bytes of a register are loaded in byte-wise reverse order. This instruction can be used to convert two, four, or eight bytes from a 'little-endian' format to a 'big-endian' format or *vice versa*.
- RLL, RLLG – Rotate Left Single Logical. These two instructions have also been made available in the ESA/390 instruction set.
- TROO – Translate one to one. Translates from one single-byte code to another single-byte code.
- TROT – Translate one to two. Translates from a single-byte code to a double-byte code.
- TRTO – Translate two to one. Translates from a double-byte code to a single-byte code.
- TRTT – Translate two to two. Translates from one double-byte code to another double-byte code.

IARV64 SYSTEM SERVICES

To obtain virtual storage above the two gigabyte line (ie above the

bar), IBM has introduced a new set of system services called the IARV64 System Services. Storage above the bar is in memory objects. A memory object can be as large as the memory limits set by the installation or as small as one megabyte. The existing GETMAIN/FREEMAIN services, STORAGE macro, CPOOL, or callable cell pool services do not work on virtual storage above the bar. To use the storage in memory objects, the program must be executing in AMODE 64 (ie SAM 64 into AMODE64).

A new limit called the MEMLIMIT has been introduced that indicates how much virtual storage above the bar each address space can use. If no MEMLIMIT is set, the system default is 0, which means that no address space can use virtual storage above the bar. The MEMLIMIT can be specified as xxxxxM (megabytes), xxxxxG (gigabytes) xxxxxT (terabytes), xxxxxP (petabytes) or NOLIMIT.

The MEMLIMIT for each address space can be set as follows:

- The new MEMLIMIT JCL keyword on the EXEC and JOB statement. The MEMLIMIT on the JOB statement overrides the MEMLIMIT value specified on the EXEC statement.
- SMFPRMxx PARMLIB member MEMLIMIT keyword.
- SMF MEMLIMIT can be updated using the SET SMF or SETSMF commands.
- MEMLIMIT set in the IEFUSI installation exit. This setting overrides all other settings.

The IARV64 services provided are as follows:

- GETSTOR – create a memory object
- DETACH – free one or more memory objects
- PAGEFIX – fix physical pages
- UNPAGEFIX – reverse a PAGEFIX operation
- PAGEOUT – perform a PAGEOUT request
- PAGEIN – perform a PAGEIN request

- DISCARDATA – Discard Data in memory objects
- CHANGEGUARD – create a guard area and a usable area in the memory object
- LIST – List the memory objects.

The scope of these services is based on problem state, key 8-F programs, and supervisor state or key 0-7 programs.

The following example creates a 1-megabyte memory object:

```
IARV64   REQUEST=GETSTOR,
         SEGMENTS=ONESEG,
         USERTLM=UTOKEN,
         ORIGIN=V64ADDR,
         COND-YES

ONESEG   DC   ADL8(1)
UTOKEN   DC   ADL8(1)
V64ADDR  DS   AD
```

Note the use of the AD 8-byte doubleword-aligned address constants.

64-BIT STANDARD LINKAGE CONVENTION

New 64-bit standard linkage conventions have been introduced. The linkage conventions that are defined are as follows.

Standard 72-byte save area

This type of save area would be used if the target program does not change the high-order halves of the General Purpose Registers. The GPRs are saved with the STM instruction.

Format 4 save area (F4SA)

This type of save area would be used if the target program changes the high-order halves of the General Purpose Registers. The length is 144 bytes. The GPRs are saved with the STMG instruction.

Its format is as follows:

- 0 – used by language translators
- 1 – ‘F4SA’
- 2-3 – 64-bit GPR14 (Return Address)
- 4-5 – 64- bit GPR 15
- 6-31 – 64-bit GPRs 0-12
- 32-33 – address of the previous save area (stored by the calling program)
- 34-35 – address of the next save area (stored by the target program).

Format 5 save area (F5SA)

This type of save area would be used if the target program is AMODE64 and the caller still passes a 72-byte savearea. The length is 216 bytes. The low-order halves of the registers are saved in the caller’s savearea with the STM instruction and the high-order halves of the registers are saved with the STMH instruction in the target program’s Format 5 save area (F5SA), words 36-53.

Its format is as follows:

- 0 – used by language translators
- 1 – ‘F4SA’
- 2-3 – 64-bit GPR14 (Return Address)
- 4-5 – 64- bit GPR 15
- 6-31 – 64-bit GPRs 0-12
- 32-33 – address of the previous save area (stored by the calling program)
- 36-53 – address of the next save area (stored by the target program)

- 36-53 – high-order halves of GPRs 0-15.

Format 6 save area (F6SA)

This type of save area would be used if a primary mode program saves the registers on the linkage stack and calls another program that requires a 36-word save area.

Its format is as follows:

- 0 – used by language translators
- 1 – 'F4SA'
- 2-3 – 64-bit GPR14 (Return Address)
- 4-5 – 64-bit GPR 15
- 6-31 – 64-bit GPRs 0-12
- 32-33 – address of the previous save area (stored by the calling program)
- 34-35 – address of the next save area (stored by the target program).

CONTROL BLOCK INFORMATION

The following control blocks have been updated with some interesting 64-bit information.

CVT

Flag CVTOSLV3, byte 3 of CVTOSLVL has the following equates defined:

```
CVTZOS          EQU X'20' – z/OS V1R1
CVTZOS_010100   EQU X'20' – z/OS V1R1
CVTZOS_V1R1     EQU X'20' – z/OS V1R1
CVTZOS_010200   EQU X'10' – z/OS V1R2
```

CVTZOS_V1R21 EQU X'10' – z/OS V1R2

CVT64 EQU X'10' – 64-bit Virtual.

Field CVTOSLVL is part of the programming interface information.

RAX (RSM Address Space Block Extension)

The following fields are part of the programming interface information:

- RAXLVMEMLIM – address Space Memory Limit in MB.
- RAXLVABYTES – number of bytes allocated from large virtual memory in memory objects.
- RAXLVNMOMB – number of memory objects allocated.

HIGH LEVEL ASSEMBLER RELEASE 4

I would like to mention a couple of zSeries support features that have been incorporated into HLASM Release 4.

AMODE changes

The AMODE instruction now supports the following directives:

- ANY31 – the control section or entry point is not sensitive to whether it is entered in AMODE 24 or AMODE 31. ANY31 is equivalent to ANY.
- 64 – specifies that 64-bit addressing mode is to be associated with a control section or entry point.

RMODE changes

The RMODE instruction now supports the following directives:

- 31 – specifies that a residence mode of either 24 or 31 is to be associated with the control section: that is, the control section can be resident above or below the 16-megabyte line.

- 64 – specifies that a residence mode of 64 is to be associated with the control section.

Address constant

A new 8-byte address constant:

- AD – 8-byte doubleword-aligned address constant, for example:

```
0000000000001000   DC   AD(4096)
```

Binary Constant

A new 8-byte binary constant:

- FD – 8 byte doubleword-aligned binary integer, for example:

```
0000000000000400   DC   FD'1024'
```

R F Perretta
Systems Consultant
Millenium Computer Consultancy (UK)

Assessing programs for virtual storage memory leaks

Today, OS/390 systems run for extended lengths of time without requiring or generating an outage. As a result, continuous system operation has improved significantly when compared with 10 or 15 years ago. Applications that run in these environments need to be well behaved in order to stay up and running during these lengthy operational uptimes. In order to accomplish this, one of the classic problems that face long-running applications needs to be attended to. The problem referred to is the potential for an application virtual storage shortage caused by undetected memory leaks.

Memory leaks typically occur as a result of recurring storage

GETMAINs or STORAGE OBTAIN operations that do not have a corresponding FREEMAIN or STORAGE RELEASE. Given enough of these of a sufficient size an application's virtual storage region can become completely allocated. Requests for additional dynamic storage result in the traditional x78 abends.

The GFSDATAA program provided with this article offers a method of assessing dynamically acquired virtual storage usage by an application to determine the potential for memory leaks. In order to properly use GFSDATAA, you must make use of three available OS/390 components:

1. DIAG*xx* PARMLIB members
2. GTF GFS trace
3. IPCS GTFTRACE output formatting.

DIAGXX PARMLIB MEMBERS

The DIAG*xx* PARMLIB members are used to activate and deactivate virtual storage tracing. The powerful benefit of using this feature is that it can capture GETMAIN/FREEMAIN requests (SVC and branch enter calls) and STORAGE OBTAIN/RELEASE requests. The information provided with the DIAG*xx* PARMLIB member can indicate which address space should be monitored (either by jobname or ASID) and the characteristics of storage get/free events that should be captured. For example, you can request trace records for only certain subpools, storage keys, length, etc – the *OS/390 MVS Initialization and Tuning Reference* manual provides details on the DIAG*xx* PARMLIB member. A sample usage might look something like:

```
VSM TRACE GETFREE(ON) SUBPOOL(1Ø-25,231-233,253) DATA(ALL)
JOBNAME(LONGAPP1)
```

This will cause GFS (Get Free Storage) tracing to be activated for jobname LONGAPP1. Trace records will be produced for GET/FREE requests in subpools 10-25, 231-233, and 253. Maximum trace information will be captured (DATA(ALL)).

If the above parameter information were located in a PARMLIB

dataset member DIAG99, it would be activated using the following operator command:

```
SET DIAG=99
```

IBM provides a default DIAG00 PARMLIB member that can be used to turn tracing off. When you have captured the trace information you desire, tracing can be disabled with this command:

```
SET DIAG=ØØ
```

Ensure that the DIAG00 member contains the following:

```
VSM TRACE GETFREE(OFF)
```

GTF GFS TRACE

GTF is used to capture the GFS trace records produced with VSM TRACE GETFREE(ON). The following GTF control input should be supplied in the GTF SYSLIB dataset prior to starting the GTF started task:

```
TRACE=USRP
USR=(F65)
```

This will indicate that GTF should capture GFS trace records produced as a result of VSM trace activation with a SET DIAG=xx operator command. If SYS1.PARMLIB member GFSTRACE contained the above GTF control cards, a sample GTF procedure to initiate GFS tracing would look something like:

```
//GTF     PROC MEMBER=GFSTRACE
//IEFPROC EXEC PGM=AHLGTF,PARM='MODE=EXT,DEBUG=NO,TIME=YES',
//  TIME=1440,REGION=2880K
//IEFRDER DD   DSNAME=gfs.trace.dataset,UNIT=SYSDA,SPACE=(TRK,2Ø),
//             DISP=(NEW,KEEP)
//SYSLIB  DD   DSNAME=SYS1.PARMLIB(&MEMBER),DISP=SHR
```

Section 13.1 *Starting and Stopping GFS Trace* in *OS/390 MVS Diagnosis: Tools and Service Aids* provides additional information on using GTF to capture the GFS trace records.

IPCS GTFTRACE OUTPUT FORMATTING

IPCS is used to format the trace records produced from GTF. The

GFSDATAA program expects its input data to be IPCS GTFTRACE output data produced by running JCL similar to the following:

```
//IPCS     EXEC PGM=IKJEFT01,DYNAMNBR=20,REGION=1500K
//IPCSDDIR DD DSN=ipcs.DDIR,DISP=(SHR)
//DUMP     DD DSN=gfs.trace.dataset,DISP=SHR
//IPCSPRNT DD DSN=gfs.gtftrace.output,DISP=(,CATLG),UNIT=SYSDA,
//         SPACE=(TRK,(5,1)),
//         DCB=(LRECL=133,BLKSIZE=1334,DSORG=PS,RECFM=VBA)
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
  IPCS
  DROPDUMP DDNAME(DUMP)
  GTFTRACE USR(F65)                +
      DDNAME(DUMP) PRINT NOTERMINAL
  Y
  DROPDUMP DDNAME(DUMP)
  END
```

IPCS GTFTRACE output generated to the IPCSPRNT dataset will have the following general appearance:

```
IPCS PRINT LOG FOR USER USER1
_____________________________________________________________________
   **** GTFTRACE DISPLAY OPTIONS IN EFFECT ****
 USR=SEL
 **** GTF DATA COLLECTION OPTIONS IN EFFECT: ****
 USRP option
              **** GTF TRACING ENVIRONMENT ****
     Release: SP6.1.0   FMID: HBB7703   System name: TCM0
     CPU Model: 9672  Version: 86  Serial no. 126659
USRDA F65 ASCB 00EB7700            JOBN LNGAPP
Getmain Local branch(120)  Cond=No  CheckZero=No
Loc=(24,31)  Bndry=Dblwd
Return address=00E0C2D0  Amode=31  Asid=005B  Jobname=LNGAPP
Subpool=255  Key=0  Asid=005B  Jobname=LNGAPP    TCB=009FE0A8  Retcode=0
Storage address=00515488  Length=1344 X'540'
  GPR Values
      0-3  00000540  00515488  00000001  0000FF02
      4-7  009FE0A8  00000011  7F8F495C  00EB7700
      8-11 7F8F50A0  80E0C27A  00000001  00000001
     12-15 00000045  00000000  80E0C2D0  00000000
               GMT-11/07/2001 19:22:23.504066  LOC-11/07/2001
13:22:23.504066
USRDA F65 ASCB 00EB7700            JOBN LNGAPP
Freemain Local branch(120)  Cond=No  CheckZero=No
Return address=00E0CDF6  Amode=31  Asid=005B  Jobname=LNGAPP
Subpool=255  Key=0  Asid=005B  Jobname=LNGAPP    TCB=009FE0A8  Retcode=0
Storage address=00515488  Length=1344 X'540'
```

```
  GPR Values
      Ø-3  00000540  00515488  00000001  0000FFØ3
      4-7  009FEØA8  00000001  0003B5B4  00EB77ØØ
      8-11 00515488  8ØE0C27A  005154DØ  00000000
     12-15 00000000  00000000  80E0CDF6  00000000
                GMT-11/07/2001 19:22:23.504213  LOC-11/07/2001
13:22:23.504213
```

VIRTUAL STORAGE USAGE ANALYSIS

The nature of program code, especially code that will be up and running for a period of time, is to:

1 Acquire storage for long-term use.

2 Wait for something to happen.

3 Process an event (generally requiring temporary use storage).

4 Return to 2, unless the request is to terminate.

What GFSDATAA is good at analysing is storage GET/FREE requests that occur in 3 above. As a result, the recommended data capture approach would be:

- Start the long running application.
- Activate the appropriate DIAG=xx PARMLIB member.
- Start GTF GFS tracing.
- Run multiple tests into the application.
- Terminate the GTF trace and deactivate GFS tracing with the DIAG00 PARMLIB member.
- Produce the IPCS GTFTRACE output from the GTF trace dataset.
- Use GFSDATAA to examine the IPCS GTFTRACE output dataset contents.

THE GFSDATAA PROGRAM

The following is sample JCL to run the GFSDATAA program:

```
//GFSDATAA EXEC PGM=GFSDATAA
//STEPLIB  DD   DSN=load.library,DISP=SHR
//INPUT    DD   DSN=gfs.gtftrace.output,DISP=SHR
//OUTPUT   DD   SYSOUT=*
```

If GFSDATAA is able to pair up all the storage GET/FREE requests, GFSDATAA will end with a return code of 0 and produce one line of output in the OUTPUT dataset as follows:

```
ALL ALLOCATED STORAGE HAS BEEN RELEASED
```

If GFSDATAA does not find a corresponding storage free request for a prior storage get request, GFSDATAA will end with a return code of 4 and produce a report in the OUTPUT dataset containing information similar to the following:

```
POTENTIAL MEMORY LEAK DETECTED
STORAGE ADDR 00515550 LEN(00000478) NOT RELEASED FOR TRACE ENTRY
13:22:25.584172
STORAGE ADDR 00515390 LEN(000001C0) NOT RELEASED FOR TRACE ENTRY
13:22:25.586749
STORAGE ADDR 00518040 LEN(00000050) NOT RELEASED FOR TRACE ENTRY
13:22:25.657453
STORAGE ADDR 00514B88 LEN(00000478) NOT RELEASED FOR TRACE ENTRY
13:24:30.377423
STORAGE ADDR 005151D0 LEN(000001C0) NOT RELEASED FOR TRACE ENTRY
13:24:30.380545
STORAGE ADDR 00515180 LEN(00000050) NOT RELEASED FOR TRACE ENTRY
13:24:30.380628
STORAGE ADDR 00514710 LEN(00000478) NOT RELEASED FOR TRACE ENTRY
13:30:34.652209
```

Notice that the storage address, storage length, and time stamp of the trace entry are indicated. This allows you to go back into the IPCS GTFTRACE output to identify the offending trace entry and then initiate corrective action.

On rare occasions, the GTF GFS trace may have captured a storage free request that didn't have a prior storage get trace record. Make a note and be aware that the most likely cause of this is that the application being examined acquired a storage block prior to the GFS trace being activated. In this case, GFSDATAA will also end with a return code of 4 and produce an output record in the OUTPUT dataset similar to:

```
GFS TRACE DOES NOT CONTAIN OBTAIN REFERENCE FOR FREED STORAGE ADDRESS
00514420
```

CONCLUSION

GFSDATAA is a very useful tool for assessing virtual storage usage in application code. If you will be creating long-running application code, you should consider using a tool like GFSDATAA prior to releasing the application into production use. It will allow you to examine virtual storage usage to determine whether anything unexpected is occurring with dynamic storage acquisition or release.

GFSDATAA.ASM

```
*************************************************************************
*                                                                       *
*   The GFSDATAA (Get Free Storage DATA Analysis) program is used       *
*   to examine IPCS GTFTRACE formatted output from a GTF GFS trace.     *
*   The purpose of the program is to try and determine whether there    *
*   are any virtual storage memory leaks in the program code that       *
*   has been traced.                                                    *
*                                                                       *
*   Three standard OS/390 components are used in obtaining the          *
*   input information used by GFSDATAA.  The DIAGxx PARMLIB member      *
*   is used to activate/deactive the trace as well as to indicate       *
*   which address spaces data is to be captured for (see OS/390         *
*   Initialization and Tuning Reference for details).  GTF is used      *
*   to capture the GFS trace records that get produced when GFS         *
*   tracing has been enabled by a SET DIAG=xx command.  An IPCS         *
*   GTFTRACE is used to produce the formatted output listing that       *
*   is used as input data to the GFSDATAA utility.                      *
*                                                                       *
*   This program should be linkedited into a target load library        *
*   using the following linkedit control cards:                         *
*                                                                       *
*        INCLUDE OBJECT(GFSDATAA)                                       *
*        ENTRY   GFSDATAA                                               *
*        NAME    GFSDATAA(R)                                            *
*                                                                       *
*   The following JCL can be used to produce a GFS analysis report      *
*   from an IPCS GTFTRACE output listing:                               *
*                                                                       *
*   //GFSDATAA EXEC PGM=GFSDATAA                                        *
*   //STEPLIB  DD   DSN=load.library,DISP=SHR                           *
*   //INPUT    DD   DSN=ipcs.gtftrace.gfs.output,DISP=SHR               *
*   //OUTPUT   DD   SYSOUT=*                                            *
*************************************************************************
GFSDATAA CSECT
GFSDATAA AMODE 31
```

```
GFSDATAA RMODE 24                    GOT SOME DCB'S
         STM   R14,R12,12(R13)       SAVE THE REGISTERS
         LR    R12,R15               COPY BASE REGISTER
         USING GFSDATAA,R12          SET ADDRESSABILITY
         LR    R11,R1                SAVE INCOMING PARM ADDRESS
         LR    R3,R13                SAVE INCOMING SAVEAREA ADDRESS
         STORAGE OBTAIN,LENGTH=WORKLEN,LOC=ANY
         LR    RØ,R1                 COPY THE ADDRESS
         LR    R14,R1                AGAIN
         LR    R2,R1                 AGAIN
         L     R1,=A(WORKLEN)        GET THE LENGTH
         XR    R15,R15               SET FILL BYTE VALUE
         MVCL  RØ,R14                CLEAR THE STORAGE
         ST    R3,4(,R2)             SAVE OLD SAVEAREA ADDRESS
         LR    R13,R2                GET NEW SAVEAREA ADDRESS
         USING WORKAREA,R13          SET ADDRESSABILITY
         LR    R1,R11                COPY PARM ADDRESS
         OPEN  (INPUT,INPUT),MODE=31 OPEN INPUT DATASET
         OPEN  (OUTPUT,OUTPUT),MODE=31 OPEN OUTPUT DATASET
EVENTLP  EQU   *
         GET   INPUT,INRECLN         GET INPUT DATA
         CLC   INREC+1(9),=C'USRDA F65' AN EVENT START RECORD?
         BE    EVENTST               YES - DETERMINE EVENT TYPE
         B     EVENTLP               GET NEXT RECORD
EVENTST  EQU   *
         GET   INPUT,INRECLN         GET INPUT DATA
         CLC   INREC+1(14),=C'Storage Obtain' STORAGE GET FUNCTION?
         BE    STORGET               YES - PROCESS GET
         CLC   INREC+1(7),=C'Getmain' STORAGE GET FUNCTION?
         BE    STORGET               YES - PROCESS GET
         CLC   INREC+1(15),=C'Storage Release' STORAGE FREE FUNCTION?
         BE    STORFREE              YES - PROCESS FREE
         CLC   INREC+1(8),=C'Freemain' STORAGE FREE FUNCTION?
         BE    STORFREE              YES - PROCESS FREE
         B     EVENTST               FIND START RECORD
STORGET  EQU   *
         GET   INPUT,INRECLN         GET INPUT DATA
         CLC   INREC+1(16),=C'Storage address=' DATA RECORD?
         BE    GETDATA               YES - RECORD WE'RE LOOKING FOR
         CLC   INREC+1(15),=C'Return address=' ASID RECORD?
         BE    GETASID               YES - RECORD WE'RE LOOKING FOR
         CLC   INREC+1(9),=C'USRDA F65' AN EVENT START RECORD?
         BE    EVENTST               YES - DETERMINE EVENT TYPE
         B     STORGET               TRY NEXT RECORD
GETASID  EQU   *
         MVC   ASIDSAVE(4),INREC+31  SAVE THE ASID
         B     STORGET               GET NEXT RECORD
GETDATA  EQU   *
*********************************************************************
*   OBTAIN STORAGE FOR A STORAGE GET TRACE ENTRY.  WE WILL CAPTURE    *
```

```
*   KEY PIECES OF INFORMATION AND SAVE IT HERE.                         *
*************************************************************************
         STORAGE OBTAIN,LENGTH=VSAELN,LOC=ANY GET ENTRY SAVE AREA
         XC    Ø(VSAELN,R1),Ø(R1)    CLEAR IT
         MVC   VSAEASID-VSAE(4,R1),ASIDSAVE COPY THE ASID
         MVC   Ø(4,R1),ALLOCPTR      CHAIN TO NEXT ENTRY
         ST    R1,ALLOCPTR           SAVE NEW CHAIN START
         MVC   DBL1(8),INREC+17      SAVE THE STORAGE ADDRESS
         TR    DBL1(8),TRTABLE1      CONVERT TO HEX CHARACTERS
         PACK  DBL2(5),DBL1(9)       PACK IT
         MVC   4(4,R1),DBL2          COPY THE STARTING ADDR
         LA    R15,INREC+34          POINT TO LENGTH
         XR    R2,R2                 CLEAR R2
         XR    R3,R3                 CLEAR R3
LENLP1   EQU   *
         CLI   Ø(R15),C' '           END OF LENGTH?
         BE    DONELEN1              YES - DO LENGTH END STUFF
         MH    R2,=H'1Ø'             MULTIPLY CURRENT BASE BY 1Ø
         ICM   R3,B'ØØØ1',Ø(R15)     MOVE IN CURRENT DIGIT
         N     R3,=X'ØØØØØØØF'       TURN OFF FIRST NIBBLE
         AR    R2,R3                 ADD TO LENGTH
         LA    R15,1(,R15)           POINT TO NEXT BYTE
         B     LENLP1                CHECK NEXT BYTE
DONELEN1 EQU   *
         ST    R2,12(,R1)            SAVE LENGTH
         L     R15,4(,R1)            GET START ADDRESS
         AR    R15,R2                SET END ...
         BCTR  R15,Ø                    ADDRESS
         ST    R15,8(,R1)            SAVE ENDING ADDRESS
TIMEGET  EQU   *
         GET   INPUT,INRECLN         GET INPUT DATA
         CLC   INREC+1(9),=C'USRDA F65' AN EVENT START RECORD?
         BE    EVENTST               YES - DETERMINE EVENT TYPE
         CLC   INREC+16(4),=C'GMT-'  TIME STAMP RECORD?
         BNE   TIMEGET               NO - CHECK NEXT RECORD
         L     R1,ALLOCPTR           GET CURRENT ENTRY ADDRESS
         MVC   16(15,R1),INREC+63    SAVE TIME STAMP
         B     EVENTLP               CHECK FOR NEXT EVENT
*************************************************************************
STORFREE EQU   *
*************************************************************************
*   WE HAVE DETECTED A STORAGE FREE TRACE EVENT.  CAPTURE ASID,       *
*   STORAGE ADDRESS, AND LENGTH INFORMATION AND DETERMINE IF THERE    *
*   IS A STORAGE GET ENTRY ON THE CHAIN THAT MATCHES THIS FREE        *
*   REQUEST.                                                          *
*************************************************************************
         GET   INPUT,INRECLN         GET INPUT DATA
         CLC   INREC+1(16),=C'Storage address=' DATA RECORD?
         BE    FREEDATA              YES - RECORD WE'RE LOOKING FOR
         CLC   INREC+1(15),=C'Return address=' ASID RECORD?
```

```
         BE    FREEASID              YES - RECORD WE'RE LOOKING FOR
         CLC   INREC+1(9),=C'USRDA F65' AN EVENT START RECORD?
         BE    EVENTST               YES - DETERMINE EVENT TYPE
         B     STORFREE              TRY NEXT RECORD
FREEASID EQU   *
         MVC   ASIDSAVE(4),INREC+31  SAVE THE ASID
         B     STORFREE              GET NEXT RECORD
FREEDATA EQU   *
         MVC   DBL1(8),INREC+17      SAVE THE STORAGE ADDRESS
         TR    DBL1(8),TRTABLE1      CONVERT TO HEX CHARACTERS
         PACK  DBL2(5),DBL1(9)       PACK IT
         LA    R15,INREC+34          POINT TO LENGTH
         XR    R2,R2                 CLEAR R2
         XR    R3,R3                 CLEAR R3
LENLP2   EQU   *
         CLI   Ø(R15),C' '           END OF LENGTH?
         BE    DONELEN2              YES - DO LENGTH END STUFF
         MH    R2,=H'1Ø'             MULTIPLY CURRENT BASE BY 1Ø
         ICM   R3,B'ØØØ1',Ø(R15)     MOVE IN CURRENT DIGIT
         N     R3,=X'ØØØØØØØF'       TURN OFF FIRST NIBBLE
         AR    R2,R3                 ADD TO LENGTH
         LA    R15,1(,R15)           POINT TO NEXT BYTE
         B     LENLP2                CHECK NEXT BYTE
DONELEN2 EQU   *
*********************************************************************
*   AT THIS POINT, DBL2 CONTAINS THE ADDRESS OF THE FREED BLOCK OF  *
*   STORAGE AND R2 CONTAINS THE FREED BLOCK LENGTH.  RUN THROUGH THE*
*   GET STORAGE CHAIN LOOKING FOR A BLOCK THAT CONTAINS THIS        *
*   FREE AREA ADDRESS.                                              *
*********************************************************************
         L     R4,ALLOCPTR           GET CHAIN START ADDRESS
         LA    R5,ALLOCPTR           KEEP ADDR OF SAVE AREA
ALLOCCHK EQU   *
         LTR   R4,R4                 AN ALLOC ENTRY?
         BZ    NOMATCH               NO - ISSUE A MESSAGE
         CLC   DBL2(4),4(R4)         ADDRESS TOO LOW?
         BL    NEXTENT               YES - TRY NEXT ENTRY
         CLC   DBL2(4),8(R4)         ADDRESS TOO HIGH?
         BH    NEXTENT               YES - TRY NEXT ENTRY
         CLC   VSAEASID-VSAE(4,R4),ASIDSAVE SAME ASID?
         BNE   NEXTENT               NO - ADDR IS IN RANGE, ASID DIFF
         L     R1,12(,R4)            GET LENGTH
         CR    R2,R1                 A LENGTH MATCH?
         BNE   NOTALL                NO - WE'RE FREEING THE WHOLE CHUNK
*********************************************************************
*   WE'VE DETECTED AN ALLOCATED STORAGE BLOCK CONTAINING THE FREE   *
*   ADDRESS.  THE ENTIRE ENTRY IS BEING FREED.  REMOVE THE ENTRY    *
*   FROM THE CHAIN AND PROCESS THE NEXT EVENT.                      *
*********************************************************************
         MVC   Ø(4,R5),Ø(R4)         MAINTAIN CHAIN INTEGRITY
```

```
         STORAGE RELEASE,LENGTH=VSAELN,ADDR=(R4)
         B     EVENTLP               CHECK FOR NEXT EVENT
NOTALL   EQU   *
         CLC   4(4,R4),DBL2          FREED CHUNK AT START?
         BE    ATSTART               YES - THIS IS THE EASIEST
         L     R15,8(,R4)            GET ENDING ADDR
         LA    R15,1(,R15)           ADD ONE
         SR    R15,R2                SUBTRACT THE LENGTH
         L     R1,DBL2               GET START ADDR
         CR    R15,R1                FREED CHUNK AT END?
         BE    ATEND                 YES - ALMOST AS EASY AS AT START
INMIDDLE EQU   *
*********************************************************************
*   WE'VE DETECTED AN ALLOCATED STORAGE BLOCK CONTAINING THE FREE     *
*   ADDRESS.  THE ADDRESS AND LENGTH INDICATE A STORAGE FREE REQUEST  *
*   IN THE MIDDLE OF AN EXISTING GET BUFFER.  WE NEED TO BREAK UP     *
*   THE GET ENTRY INTO TWO SEPARATE BLOCKS FOR FUTURE REFERENCE.      *
*********************************************************************
         STORAGE OBTAIN,LENGTH=VSAELN,LOC=ANY
         XC    Ø(VSAELN,R1),Ø(R1)    CLEAR IT
         LR    R5,R1                 SAVE STORAGE ADDRESS
         MVC   Ø(VSAELN,R5),Ø(R4)    COPY CURRENT ENTRY
         ST    R5,Ø(,R4)             ADD NEW ENTRY INTO CHAIN
         L     R15,DBL2              GET FREE BLOCK ADDR
         S     R15,4(,R4)            SUBTRACT BLOCK START ADDR
         ST    R15,12(,R4)           SAVE NEW LENGTH
         L     R1,4(,R4)             GET START ADDR
         AR    R1,R15                ADD LENGTH
         BCTR  R1,Ø                  SET END ADDR
         ST    R1,8(,R4)             SAVE NEW END ADDR
         L     R15,DBL2              GET START ADDR FOR NEW ENTRY
         ST    R15,4(,R5)            SAVE IN ENTRY
         L     R1,8(,R5)             GET END ADDR
         LA    R1,1(,R1)             ADD ONE
         SR    R1,R15                GET LENGTH
         ST    R1,12(,R5)            SAVE LENGTH OF NE ENTRY
         B     EVENTLP               CHECK FOR NEXT EVENT
ATSTART  EQU   *
*********************************************************************
*   WE'VE DETECTED AN ALLOCATED STORAGE BLOCK CONTAINING THE FREE     *
*   ADDRESS.  THE ADDRESS AND LENGTH INDICATE THAT THE FREE REQUEST   *
*   IS NOT FOR AN ENTIRE GET ENTRY, BUT IT IS RIGHT AT THE START OF   *
*   AN EXISTING ENTRY.  ALTER THE STARTING ADDRESS OF THE EXISTING    *
*   ENTRY AND ADJUST ITS LENGTH ACCORDINGLY.                          *
*********************************************************************
         L     R15,4(,R4)            GET START ADDR
         AR    R15,R2                ADD LENGTH
         ST    R15,4(,R4)            SAVE NEW START ADDR
         L     R15,12(,R4)           GET LENGTH
         SR    R15,R2                SUBTRACT FREE BLOCK LENGTH
```

```
         ST    R15,12(,R4)           SAVE NEW BLOCK LENGTH
         B     EVENTLP               CHECK FOR NEXT EVENT
ATEND    EQU   *
*********************************************************************
*   WE'VE DETECTED AN ALLOCATED STORAGE BLOCK CONTAINING THE FREE     *
*   ADDRESS.  THE ADDRESS AND LENGTH INDICATE THAT THE FREE REQUEST   *
*   IS NOT FOR AN ENTIRE GET ENTRY, BUT IT IS RIGHT AT THE END OF     *
*   AN EXISTING ENTRY.  ADJUST THE LENGTH OF THE EXISTING ENTRY       *
*   ACCORDINGLY.                                                      *
*********************************************************************
         BCTR  R15,Ø                 REDUCE ADDR BY ONE
         ST    R15,8(,R4)            SAVE NEW END ADDR
         L     R15,12(,R4)           GET LENGTH
         SR    R15,R2                SUBTRACT FREE BLOCK LENGTH
         ST    R15,12(,R4)           SAVE NEW BLOCK LENGTH
         B     EVENTLP               CHECK FOR NEXT EVENT
NEXTENT  EQU   *
         LR    R5,R4                 SAVE THIS ENTRY ADDRESS
         L     R4,Ø(,R4)             GET ADDR OF NEXT CHAIN ENTRY
         B     ALLOCCHK              GO CHECK IT OUT
NOMATCH  EQU   *
         MVC   OUTREC(8Ø),OUTREC2    MOVE IN OUTPUT RECORD
         MVC   OUTREC+7Ø(8),INREC+17 MOVE IN STORAGE ADDRESS
         PUT   OUTPUT,OUTREC         WRITE THE MESSAGE
         OI    FLAG,FREEBLK          SET UNPAIRED FREE BLOCK FLAG
         B     EVENTLP               CHECK FOR NEXT EVENT
*********************************************************************
ALLDONE  EQU   *
         CLC   ALLOCPTR(4),=F'Ø'     ANY EXISTING STORAGE REFERENCES?
         BNE   MEMLEAK               YES - MAY INDICATE MEMORY LEAK
         PUT   OUTPUT,OUTREC1        WRITE MESSAGE
         TM    FLAG,FREEBLK          UNPAIRED FREE BLOCK ENCOUNTERED?
         BO    RETURNØ4              YES - RETURN RC=4
         B     RETURN                WE'RE DONE
*********************************************************************
MEMLEAK  EQU   *
*********************************************************************
*   AT THIS POINT, WE HAVE DETERMINED THAT STORAGE GET REQUESTS IN    *
*   THE TRACE OUTPUT DATA DO NOT HAVE CORRESPONDING FREE REQUESTS.    *
*                                                                     *
*   PRODUCE AN OUTPUT RECORD FOR EACH ENTRY ON THE CHAIN.             *
*********************************************************************
         PUT   OUTPUT,OUTREC3        WRITE MESSAGE
*********************************************************************
*   PROCESS THE CHAIN FROM END TO START SO THAT THE OUTPUT RECORDS    *
*   ARE PRODUCED IN TIMESTAMP ASCENDING SEQUENCE.                     *
*********************************************************************
LEAK     EQU   *
         L     R4,ALLOCPTR           GET STARTING ENTRY ADDR
         LA    R5,ALLOCPTR           SAVE POINTER ADDR
```

```
LEAKLP   EQU   *
         CLC   Ø(4,R4),=F'Ø'          LAST ENTRY?
         BE    LASTENT                YES - PROCESS LAST ENTRY
         LR    R5,R4                  SAVE PREV ADDR
         L     R4,Ø(,R4)              POINT TO NEXT ENTRY
         B     LEAKLP                 CHECK IT OUT
LASTENT  EQU   *
         XC    Ø(4,R5),Ø(R5)          CLEAR POINTER
         MVC   DBL2(4),4(R4)          COPY BLOCK ADDRESS
         UNPK  DBL1(9),DBL2(5)        UNPACK THE ADDRESS
         NC    DBL1(8),=8X'ØF'        TURN OFF HIGH NIBBLE
         TR    DBL1(8),=C'Ø123456789ABCDEF'
         MVC   OUTREC(8Ø),OUTREC4     MOVE IN MESSAGE MODEL
         MVC   OUTREC+13(8),DBL1      MOVE IN THE ADDRESS
         MVC   DBL2(4),12(R4)         COPY BLOCK LENGTH
         UNPK  DBL1(9),DBL2(5)        UNPACK THE LENGTH
         NC    DBL1(8),=8X'ØF'        TURN OFF HIGH NIBBLE
         TR    DBL1(8),=C'Ø123456789ABCDEF'
         MVC   OUTREC+26(8),DBL1      MOVE IN THE LENGTHS
         MVC   OUTREC+65(15),16(R4)   MOVE IN TIMESTAMP
         PUT   OUTPUT,OUTREC          WRITE THE MESSAGE
         STORAGE RELEASE,LENGTH=VSAELN,ADDR=(R4)
         CLC   ALLOCPTR(4),=F'Ø'      LAST ENTRY?
         BE    RETURNØ4               YES - WE'RE DONE
         B     LEAK                   TRY ANOTHER TIME
*********************************************************************
RETURN   EQU   *
         CLOSE (INPUT),MODE=31        CLOSE INPUT DATASET
         CLOSE (OUTPUT),MODE=31       CLOSE OUTPUT DATASET
         LR    R1,R13                 COPY STORAGE ADDRESS
         L     R3,4(,R1)              GET RETURN SAVEAREA ADDRESS
         STORAGE RELEASE,LENGTH=WORKLEN,ADDR=(R1)
         LR    R13,R3                 RESTORE RETURN SAVEAREA ADDRESS
         LM    R14,R12,12(R13)        RESTORE THE REGISTERS
         XR    R15,R15                SET RETURN CODE
         BR    R14                    RETURN
*********************************************************************
RETURNØ4 EQU   *
         CLOSE (INPUT),MODE=31        CLOSE INPUT DATASET
         CLOSE (OUTPUT),MODE=31       CLOSE OUTPUT DATASET
         LR    R1,R13                 COPY STORAGE ADDRESS
         L     R3,4(,R1)              GET RETURN SAVEAREA ADDRESS
         STORAGE RELEASE,LENGTH=WORKLEN,ADDR=(R1)
         LR    R13,R3                 RESTORE RETURN SAVEAREA ADDRESS
         LM    R14,R12,12(R13)        RESTORE THE REGISTERS
         LA    R15,4                  SET RETURN CODE
         BR    R14                    RETURN
*********************************************************************
INPUT    DCB   MACRF=(GM),DDNAME=INPUT,DSORG=PS,EODAD=ALLDONE
OUTPUT   DCB   MACRF=(PM),DDNAME=OUTPUT,LRECL=8Ø,DSORG=PS
```

```
*********************************************************************
OUTREC1  DC    CL8Ø'ALL ALLOCATED STORAGE HAS BEEN RELEASED'
OUTREC2  DC    C'GFS TRACE DOES NOT CONTAIN OBTAIN REFERENCE FOR'
         DC    CL(8Ø-L'OUTREC2)' FREED STORAGE ADDRESS XXXXXXXX'
OUTREC3  DC    CL8Ø'POTENTIAL MEMORY LEAK DETECTED'
OUTREC4  DC    C'STORAGE ADDR XXXXXXXX LEN(xxxxxxxx) NOT RELEASED FOR'
         DC    CL(8Ø-L'OUTREC2)' TRACE ENTRY '
*********************************************************************
TRTABLE1 DC    256X'ØØ'
         ORG   TRTABLE1+C'A'
         DC    X'ØA0BØC0D0E0F'
         ORG   TRTABLE1+C'Ø'
         DC    X'ØØØ102Ø3Ø4Ø5Ø6Ø7Ø8Ø9'
         ORG   ,
*********************************************************************
WORKAREA DSECT
SAVEAREA DS    18F
INPARM   DS    F
ALLOCPTR DS    F
ASIDSAVE DS    CL4
SPSAVE   DS    CL4
KEYSAVE  DS    CL2
DBL1     DS    2D
DBL2     DS    2D
INRECLN  DS    CL4
INREC    DS    CL133
OUTREC   DS    CL8Ø
FLAG     DS    CL1
FREEBLK  EQU   X'8Ø'
WORKLEN  EQU   *-WORKAREA
VSAE     DSECT                    VIRTUAL STORAGE ALLOCATION ENTRY
VSAENXT  DS    F                  ADDR OF NEXT ENTRY
VSAESTRT DS    F                  VIRTUAL ADDR OF STORAGE BLOCK
VSAEEND  DS    F                  ENDING ADDR OF STORAGE BLOCK
VSAELEN  DS    F                  LENGTH OF THIS STORAGE BLOCK
VSAETIME DS    CL16               TIMESTAMP OF THIS TRACE ENTRY
VSAEASID DS    CL4                ASID
VSAERSV1 DS    CL12               RESERVED
VSAELN   EQU   *-VSAE
         $REQU
         END
```

Automating the defrag process and preparing user-friendly reports

THE NEED FOR DEFRAG OPERATIONS

We have frequently been through situations like this: there is enough space on a single disk for a new file allocation, but, because of high fragmentation, DADSM cannot manage the new allocations. This is because of the nature of allocation algorithms and the frequent creation, extension, and deletion of datasets causes the free space on DASD volumes to become fragmented. This results in inefficient use of DASD storage space, an increase in space-related abends, performance degradation caused by excessive DASD arm movement, and an increase in the time required for functions that are related to Direct Access Device Space Management (DADSM).

So the defrag operation in this situation relieves the space-related problems, consolidating the free space on a volume to help prevent out-of-space abends on new allocations. Defrag operations relocate dataset extents on a DASD volume to reduce or eliminate free-space fragmentation. In summary, the objective of the defrag is to create the most contiguous space possible on the disk.

THE UTILITY DEFRAGJ

The utility that I developed automates the defragmentation process and prepares user-friendly reports for the systems programmers. Along with three REXX programs and one edit macro, the utility uses two JCL files, two JES2 procedures, and two SYSIN datasets.

It generates a report showing before and after statistics for each defrag operation and sends it to specific e-mail addresses in a TCP/IP network via SMTP. In addition, it maintains a cumulative statistics report to keep track of the number of defrag operations

and disk volumes involved. Statistics in this report are aimed at identifying the problematic disk volumes that have been defragmented most often, and which may therefore need special attention.

The algorithm used by the utility is based on the defragmentation index value. This index varies between 0 and 999. Lower fragmentation index values represent larger pieces of free space on the volume; therefore, low fragmentation index values are better than high values. The utility will defragment only those disks in need of defragmentation

DCOLLECT records are read in the utility for all volumes that are candidates for defrag, and those volumes having an index of 250 or higher are chosen for the defrag process. You can specify a different fragmentation index in the job DEFRAGJ to meet your needs. Note that, if you keep it too low, you may have too many disks to defrag and this may affect your system's performance.

On the other hand, if a disk's fragmentation index is quite high, it doesn't necessarily mean that the disk needs the defrag process. For example, if a disk has just three cylinders left and the largest free extent is two cylinders, then its fragmentation index will be high. If we defragment it, actually we wouldn't gain too much. For this reason, free space percentage will be the second criterion in selecting defrag-candidate volumes. This is 15% in the utility; however, you can change it in step PASO4 of the JCL DEFRAGJ.

DETERMINING THE FRAGMENTATION INDEX

To determine the fragmentation index of a volume without actually performing the DEFRAG operation, code the NORUN parameter on the EXEC statement of the ADRDSSU program. In addition to the fragmentation index, the number of free cylinders, the number of free tracks, the number of free extents, the largest free extent size, and the percentage of free space on the volume will be listed:

```
//DEFRAG   EXEC PGM=ADRDSSU,PARM='TYPRUN=NORUN'
//SYSPRINT DD   SYSOUT=A
```

```
//PRIMARY  DD   VOL=SER=PEX1Ø1,UNIT=339Ø,DISP=OLD
//SYSIN    DD   *
   DEFRAG DDNAME(PRIMARY)
```

Another easy way to figure it out is to use the Option 2.1 (DASD) of the ISMF application. You can observe all space characteristics of the disk volumes including the fragmentation index value.

Example:

```
LINE       VOLUME FREE       %      ALLOC      FRAG   LARGEST    FREE
OPERATOR   SERIAL SPACE      FREE   SPACE      INDEX  EXTENT     EXTENTS
--(1)----  -(2)-- --(3)--    (4)-    --(5)--   -(6)-  --(7)--    --(8)--
           PSE1Ø1   1Ø2Ø229    37    1751271    115     854885      1Ø2
           PSE1Ø2    681573    25    2Ø89927    194     381818       9Ø
           PSE1Ø3    552474    2Ø    2219Ø26    238     124395       47
           PSE1Ø4   1151Ø489    42    1621Ø11    241     3791Ø7      129
           PSE1Ø5    859256    31    1912244    249     4Ø2292      127
---------  ------ ----------  BOTTOM  OF  DATA  ----------  ------  ----
```

WHEN TO RUN THE DEFRAGJ FUNCTION

The defrag command of the program ADRDSSU locks the VTOC (via RESERVE) and VVDS, if it exists, on the volume. The DEFRAG function that uses the DEFRAGJ job also serializes on datasets via ENQ or dynamic allocation. These activities might cause excessive wait time for other jobs to update the VTOC. Therefore, times of low system activity are best for DEFRAGJ runs. In our data centre we schedule it for midnight on a daily basis.

HOW TO SET UP THE DEFRAGJ UTILITY

Execute the job DEFRAGJ1 to create some of the utility's datasets. Place the procedures (DEFRAGP1 and DEFRAGPB) into one of your JES2 procedure libraries and the REXX EXECs and the edit macro (DEFRAGR1, DEFRAGR2, DEFRAGR3, and DEFRAGM) into one of your SYSPROC libraries. Do not forget in the procedure members to update the library name in the SYSPROC DD statement with the library you've chosen. Enter the names of datasets in the exclude sysin dataset (EXP.SMS.DEFRAG.SYSIN.EXCLUDE) so that the DEFRAG

doesn't move them. The final thing to do is schedule the job DEFRAGJ on a daily/weekly basis, according to your requirements, via a scheduler product.

Additional notes:

1 DFSMSdss automatically excludes and does not relocate some of the datasets, eg user catalogs or RACF control datasets, so you need not exclude them in the defrag operation. (Please refer to *DFSMSdss Storage Administration* book to get a complete list of excluded datasets.) On the other hand, you have to exclude system datasets that are opened and are being accessed without an enqueue; for example procedure libraries can be excluded from the defrag.

2 You can specify the CONSOLIDATE option in the SYSIN dataset (EXP.SMS.DEFRAG.SYSIN). In this case, DEFRAG attempts to combine the dataset extents into a single extent when possible.

Dataset nomenclature used in the utility:

- EXP.SMS.CNTL – PO dataset which consists of REXX EXECs/edit macro/JCL
- EXP.SMS.DEFRAG.DCOLLECT – Dcollect dataset
- EXP.SMS.DEFRAG.REPORT – defrag-candidate disks report dataset
- EXP.SMS.DEFRAG.STATS – daily defrag statistics report dataset (cumulative)
- EXP.SMS.DEFRAG.SUMMARY – daily defrag summary report dataset
- EXP.SMS.DEFRAG.SUMMARY.GDG – daily defrag summary report (gdg dataset)
- EXP.SMS.DEFRAG.SYSIN – defrag sysin1 (includes fixed control statements)

- EXP.SMS.DEFRAG.SYSIN.EXCLUDE – defrag sysin2 (includes datasets to be excluded from defrag).

SAMPLE REPORTS GENERATED BY DEFRAGJ

Defrag detailed report for each disk volume (spool output, SYSTSPRT):

```
******************************** TOP OF DATA ***************************

DEFRAG STATISTICS FOR VOLUME - PSE1Ø3 -

                                    BEFORE      AFTER
                                    ------      ------
FREE CYLINDERS                      ØØØ642      ØØØ656
FREE TRACKS                         ØØØ26Ø      ØØØ05Ø
FREE EXTENTS                        ØØØØ36      ØØØØØ3
LARGEST FREE EXTENT (CYL,TRK)  ØØØ154,ØØØ2 ØØØ47Ø,ØØ24
FRAGMENTATION INDEX                  Ø.239       Ø.Ø95

PERCENT FREE SPACE                      19
DATASET EXTENTS RELOCATED             ØØØ22Ø
TRACKS RELOCATED                      ØØ4939
****************************** BOTTOM OF DATA ************************
```

Defrag summary report (dataset, EXP.SMS.DEFRAG.SUMMARY):

```
****************************** Top of Data ******************************
 SUMMARY OF DEFRAGMENTATION PROCESS IN THE LPAR -P1Ø1- 16/Ø4/Ø2 | 5:ØØam

Volume    T1        T2        T3        T4
------    ------    ------    ------    ------
PSE1Ø3    154       47Ø       36        3
PEX1Ø6    554      168Ø      1Ø1        3
```

Note: to get more information for each defragmented disk, check out the EXPSMS02 ouput in the spool:

```
LEYENDA:
========
T1: VOLUME Largest_Free_Extent (Cyl) - Before Defrag
T2: VOLUME Largest_Free_Extent (Cyl) - After  Defrag
T3: Free Extents                     - Before Defrag
T4: Free Extents                     - After  Defrag
****************************** Bottom of Data ************************
```

Defrag statistics report (dataset, EXP.SMS.DEFRAG.STATS):

```
****************************** Top of Data ***************
16/04/02 - 2 PSE103,PEX106
18/04/02 - 5 PEX111,PSE101,PEX113,PEX116,PSE102
21/04/02 - 1 PEX106
****************************** Bottom of Data *************
```

DEFRAGJ JCL

```
//EXPSMS01  JOB   MSGCLASS=X,MSGLEVEL=(1,1),NOTIFY=&SYSUID,CLASS=E
//*
//*-------------------------------------------------------------------*/
//*              AUTOMATED DEFRAG UTILITY (- Batch -)                 */
//*                                                                   */
//* Jcl           : DefragJ                                           */
//* REXX's called : DefragR1, DefragR2                                */
//* Function      : Generates an report which will consist of defrag- */
//*                 candidate disk volumes, then defragments them.    */
//*                                                                   */
//*-------------------------------------------------------------------*/
//PASO1    EXEC PGM=IDCAMS
//*-------------------------------------------------------------------*/
//* Delete Report and Dcollect dataset.                              */
//*-------------------------------------------------------------------*/
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
 DELETE EXP.SMS.DEFRAG.DCOLLECT
 DELETE EXP.SMS.DEFRAG.REPORT
//*
//*-------------------------------------------------------------------*/
//* Collect dataset information for all online volumes.              */
//*-------------------------------------------------------------------*/
//PASO2    EXEC PGM=IDCAMS
//SYSPRINT DD   SYSOUT=*
//DCOUT    DD   DSN=EXP.SMS.DEFRAG.DCOLLECT,
//         DISP=(NEW,CATLG,DELETE),
//         SPACE=(TRK,(3,1),RLSE),
//         DSORG=PS,RECFM=VB,LRECL=644
//SYSIN    DD   *
  DCOLLECT OUTFILE(DCOUT) VOLUMES(*) NODATAINFO
//*
//*-------------------------------------------------------------------*/
//* Allocate Report dataset.                                         */
//*-------------------------------------------------------------------*/
//PASO3    EXEC PGM=IEFBR14
//REPORT   DD   DSN=EXP.SMS.DEFRAG.REPORT,DISP=(NEW,CATLG),
// BLKSIZE=0,SPACE=(TRK,(3,1)),RECFM=FBA,LRECL=133,UNIT=SYSDA
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
```

```
//*
//*-------------------------------------------------------------------*/
//* DefragR1 :                                                        */
//* Build Report dataset by using Dcollect records.                   */
//* This report will hold Defrag-candidate volumes that meet the      */
//* conditions set by two parameters: Frag.index and Free space% .    */
//*                                                                   */
//* Parameters passed to the REXX DefragR1                            */
//* ====================================                              */
//* These are Defrag critera parameters.                              */
//*                                                                   */
//* 250:Fragmentation index     threshold                              */
//* 15 :Disk free space percent threshold                             */
//*                                                                   */
//* In order a disk volume to be defragmented, it must have fragm.    */
//* index of 250 or more AND Free space usage of 15% or more.         */
//*-------------------------------------------------------------------*/
//PASO4    EXEC DEFRAGP1
//DCOLIN   DD   DISP=SHR,DSN=EXP.SMS.DEFRAG.DCOLLECT
//REPORT   DD   DISP=SHR,DSN=EXP.SMS.DEFRAG.REPORT
//SYSTSIN  DD   *
PROFILE NOPREF
ISPSTART CMD(DEFRAGR1 250 15) +
BATSCRW(132) BATSCRD(27) BREDIMAX(3) BDISPMAX(99999999)
//*
//*-------------------------------------------------------------------*/
//* DefragR2 : Builds a dynamic Jcl. Job steps of this Jcl calls the  */
//*            DefragPB procedure to defrag a single disk volume.     */
//*                                                                   */
//*             Example:                                              */
//*             //EXPSMS02 JOB MSGCLASS=Z,MSGLEVEL=(1,1),TIME=1440,   */
//*             //             NOTIFY=&SYSUID,CLASS=E                 */
//*             //*                                                   */
//*             //STEP1 EXEC DEFRAGPB,DSK=PSE104,FLAG=0               */
//*             //STEP2 EXEC DEFRAGPB,DSK=PSE103,FLAG=0               */
//*             //STEP3 EXEC DEFRAGPB,DSK=PEX190,FLAG=1               */
//*                                                                   */
//*-------------------------------------------------------------------*/
//PASO5    EXEC DEFRAGP1
//SYSTSIN  DD   *
ISPSTART CMD(DEFRAGR2) +
BATSCRW(132) BATSCRD(27) BREDIMAX(3) BDISPMAX(99999999)
//*
//*------------------- End of Job DefragJ -------------------------*/
```

DEFRAGJ1 JCL

```
//EXPSMS01 JOB  MSGCLASS=X,MSGLEVEL=(1,1),CLASS=E
```

```
//*-------------------------------------------------------------------*/
//*              AUTOMATED DEFRAG UTILITY (- Batch -)                 */
//*                                                                   */
//* Jcl           : DefragJ1                                          */
//* Function      : This JCL has to be executed first to set-up the   */
//*                 "Automatic Defrag" utility datasets.              */
//*                                                                   */
//* Datasets created                                                  */
//* ================                                                  */
//* 1- Exp.Sms.Defrag.Summary.Gdg  : GDG dataset to hold daily        */
//*                                  Defrag summary report.           */
//*                                                                   */
//* 2- Exp.Sms.Defrag.Stats        : Statistics dataset to hold daily */
//*                                  Defrag operation statistics.     */
//*                                                                   */
//* 3- Exp.Sms.Defrag.Sysin        : Sysin used by Adrdssu program.   */
//* 4- Exp.Sms.Defrag.Sysin.Exclude: Sysin used by Adrdssu for        */
//*                                  excluding datasets from Defrag.  */
//*                                                                   */
//*-------------------------------------------------------------------*/
//DEFINE1  EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
      DEF  GDG (NAME(EXP.SMS.DEFRAG.SUMMARY.GDG) -
                NOEMPTY                           -
                SCRATCH                           -
                LIMIT(7))
//*-------------------------------------------------------------------*/
//DEFINE2  EXEC PGM=IEFBR14
//STATS    DD   DSN=EXP.SMS.DEFRAG.STATS,DISP=(,CATLG),
// DCB=(LRECL=9Ø,RECFM=FB,BLKSIZE=Ø),UNIT=SYSALLDA,SPACE=(CYL,1)
//*-------------------------------------------------------------------*/
//DEFINE3  EXEC PGM=ICEGENER
//SYSUT1   DD   *
 DEFRAG DDNAME(IN) MAXMOVE(999999,1) PASSDELAY(999) WAIT(2,2) -
        EXCLUDE(DDNAME(EXCLUDE))
//*
//SYSUT2   DD   DSN=EXP.SMS.DEFRAG.SYSIN,DISP=(,CATLG),
// DCB=(LRECL=8Ø,RECFM=F,BLKSIZE=Ø),UNIT=SYSALLDA,SPACE=(TRK,1)
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  DUMMY
//*
//*-------------------------------------------------------------------*/
//DEFINE4  EXEC PGM=ICEGENER
//SYSUT1   DD   *
 EXCLUDE(LIST(             -
              ENTER.HERE.DATA.SETS -
              ENTER.HERE.DATA.SETS -
             ))
```

```
//*
//SYSUT2   DD   DSN=EXP.SMS.DEFRAG.SYSIN.EXCLUDE,DISP=(,CATLG),
// DCB=(LRECL=8Ø,RECFM=F,BLKSIZE=Ø),UNIT=SYSALLDA,SPACE=(TRK,1)
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  DUMMY
//*
//*-------------------- End of Job DefragJ1 -------------------------*/
```

DEFRAGM EDIT MACRO

```
/* REXX */
"Isredit Macro"
/*--------------------------------------------------------------------*/
/*              AUTOMATED DEFRAG UTILITY (- Batch -)                  */
/* Edit macro   : DefragM                                             */
/* Called from  : REXX DefragR3                                       */
/* Function     : Formats the Sysprint dataset of the defrag process. */
/*                (Eliminates unnecessary records.)                   */
/*--------------------------------------------------------------------*/
"Isredit Exclude ALL"
"Isredit Find 'FREE CYLINDERS     ' ALL"
"Isredit Find 'FREE TRACKS        ' ALL"
"Isredit Find 'FREE EXTENTS       ' ALL"
"Isredit Find 'LARGEST FREE EXTENT' ALL"
"Isredit Find 'FRAGMENTATION INDEX' ALL"
"Isredit Find 'PERCENT FREE SPACE ' ALL"
"Isredit Find 'TRACKS RELOCATED   ' ALL"
"Isredit Find 'DATASET EXTENTS RELOCATED' ALL"
/*--------------------------------------------------------------------*/
/* Delete all excluded lines and save the dataset.                   */
/*--------------------------------------------------------------------*/
"Isredit Delete ALL X"
"Isredit Save"
"Isredit Cancel"
Return
/*------------------ End of Edit-macro DefragM ----------------------*/
```

DEFRAGP1 JES2 PROCEDURE

```
//DEFRAGPØ PROC
//*--------------------------------------------------------------------*/
//*              AUTOMATED DEFRAG UTILITY (- Batch -)                  */
//*                                                                    */
//* Procedure     : DefragP1                                           */
//* Called from   : Job DefragJ                                        */
//* Function      : Allocates Ispf-related datasets which are          */
//*                 necessary to be able to run REXX programs          */
```

```
//*                DefragR1 and DefragR2.                          */
//*-----------------------------------------------------------------*/
//         EXEC PGM=IKJEFT01
//STEPLIB  DD   DISP=SHR,DSN=SYS1.DGTLLIB
//SYSPROC  DD   DISP=SHR,DSN=EXP.SMS.CNTL
//ISPPLIB  DD   DISP=SHR,DSN=ISP.SISPPENU
//ISPSLIB  DD   DISP=SHR,DSN=ISP.SISPSENU
//ISPTLIB  DD   DISP=SHR,DSN=ISP.SISPTENU
//ISPMLIB  DD   DISP=SHR,DSN=ISP.SISPMENU
//         DD   DISP=SHR,DSN=ISP.SISPMENU
//ISPPROF  DD   DISP=(NEW,DELETE,DELETE),DSN=&&PROF,
// DCB=(ISP.SISPTENU),SPACE=(TRK,(1,1,1)),UNIT=SYSDA
//*
//SYSTSPRT DD   SYSOUT=(,),OUTLIM=500000
//ISPLOG   DD   SYSOUT=(,),DCB=(LRECL=125,BLKSIZE=129,RECFM=VA)
//         PEND
//*------------------- End of Proc DefragP1 ----------------------*/
```

Editor's note: this article will be concluded in the next issue.

Atalay Gul
Systems Programmer
EDS Barcelona (Spain)

Contributions for MVS Update *can be sent to Trevor Eddolls at trevore@xephon.com. A copy of our Notes for Contributors is available at www.xephon.com/nfc.*

IBM has announced Lotus Domino for z/OS V6.0, which is said to provide many significant enhancements, including improved scalability with lower CPU usage.

Lotus Domino for z/OS V6.0 is integrated with WebSphere and it implements platform statistics for CPU and storage (memory) usage that integrate RMF (TM) Monitor II information directly with the Domino statistics package, and it can utilize server monitoring via the Tivoli Analyzer for Lotus Domino, which provides health assessments based on Domino statistics, including the newly-implemented platform statistics.

Enhancements allow users to not only create selective replicas, specify document and attachment size limits, set scheduled replication, and leverage streaming replication, but also gain improved wireless access to PDAs, pagers, and Web-enabled mobile phones.

Domino 6 provides persistent connections, improved session handling, better denial of service attack handling, and more administrative control over things like URL length and number of path segments.

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

* * *

IBM has announced Infoprint XML Extender for z/OS, which connects application output using XML data interchange format to the AFP print and presentation output system.

Infoprint XML Extender for z/OS is software that operates on XML documents to produce AFP output. It can accept either XML documents with companion XSL style sheets, or documents with XSL Formatting Objects (XSL-FO) as input into the process.

Infoprint XML Extender for z/OS complements and extends support in Print Services Facility (PSF) V3.3 for formatting XML data with AFP page layouts.

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

* * *

UMX Technologies has announced Mainframe in a Box, a small to medium-sized mainframe running on a specially designed Intel-based UMX Server using Microsoft Windows 2000 or XP as the GUI.

The installed software mainframe is UMX Virtual Mainframe V4.2 microcode engine, which functions between the IBM operating system and the common Intel-based hardware to 'virtualize' the hardware to the software.

Mainframe in a Box uses the original IBM operating system and existing applications, without a single modification. All operating systems (OS/390, z/OS, VM, z/VM, and VSE) and PL/I, CICS, IMS, COBOL, and DB2 applications run on this new mainframe.

PCI add-in cards support ESCON and Parallel Channel extension technologies to provide mainframe connectivity.

For further information contact:
UMX Technologies, Kruislaan 400 NL-1098 SM, Amsterdam, The Netherlands.
Tel: (+31)20 888 4044.
URL: http://www.umxtech.com/index0.html.