In this issue

3  A simple ISPF productivity aid
6  Execute program with extended parameter
12 System layout verification tool
21 Comparing two files
31 WLM postprocessing made easy
39 Locating strings in files
51 Boosting VSAM performance with SMB
70 BPXMTEXT utility
72 Subscribing and contributing to MVS Update

© Xephon Inc 2005
MVS Update

Published by
Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA
Phone: 214-340-5690
Fax: 214-341-7081

Editor
Trevor Eddolls
E-mail: trevore@xephon.com

Publisher
Colin Smith
E-mail: info@xephon.com

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

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

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

© Xephon Inc 2005. All rights reserved. None of the text in this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior permission of the copyright owner. Subscribers are free to copy any code reproduced in this publication for use in their own installations, but may not sell such code or incorporate it in any commercial product. No part of this publication may be used for any form of advertising, sales promotion, or publicity without the written permission of the publisher.

Printed in England.
A simple ISPF productivity aid

We are always looking for ways to work smarter and quicker. In this short article we offer a simple REXX program that does just that. Three of the most common activities that most of us perform under TSO are browsing a dataset, editing a dataset, and submitting a job. Our REXX program will help us perform all three of these functions. There are two other components besides the REXX program that have to be put into place to enable this: command table entries and the EDPANEL.

COMMAND TABLE ENTRIES

These are the entries that you need to make in the ISPF SITECMDS table. You must make the following entries in that table:

ED   2 SELECT CMD(%ED ED &ZPARM) NEWAPPL(ISR)
BR   2 SELECT CMD(%ED BR &ZPARM) NEWAPPL(ISR)
SJ   2 SELECT CMD(%ED SJ &ZPARM) NEWAPPL(ISR)

Note that all three entries are pointing at the same command – %ED.

EDPANEL

The second item that needs to be placed in your local ISPF panel library is the EDPANEL, which is shown below:

)ATTR
% TYPE(TEXT) INTENS(HIGH) ATTN(OFF) SKIP(ON)
+ TYPE(TEXT) INTENS(LOW) ATTN(OFF) SKIP(ON)
@ TYPE(INPUT) INTENS(HIGH) CAPS(ON) PADC(_) ATTN(OFF)
! TYPE(INPUT) INTENS(LOW) CAPS(ON) PADC(_) ATTN(OFF) COLOR(TURQ)
{ TYPE(TEXT) COLOR(WHITE) HILITE(REVERSE) INTENS(HIGH)
)BODY EXPAND($$)
%%{$<< Alias Settings for ED/BR/SJ Commands >> $$+$ +
%OPTION====>_ZCMD
+
+ALIAS  FULLY QUALIFIED DATASET NAME  ALIAS FULLY QUALIFIED DATASET NAME
+
@A1    !D1                          +@A18   !D18
@A2    !D2                          +@A19   !D19
@A3    !D3                          +@A20   !D20
@A4    !D4                          +@A21   !D21
This is a very straightforward panel that is used to create and maintain aliases for datasets. It allows us to associate a short name with a full dataset. Typical entries might look like the following:

**JCL** hlq.my.jcl.library  
**SAS** hlq.my.sas.library

As implemented, up to 34 entries can be created. Note that this information is saved in your ISPF profile so that it will be preserved across TSO sessions.

**ED REXX PROGRAM**

The last item that we need to consider is the REXX program itself. We have kept the code very simple and straightforward. If you invoke the program by entering BR, ED, or SJ without any arguments or with ? being the only argument, the EDPANEL panel will be displayed, so that you can maintain the aliases and their associated datasets. If you invoke it with only a single passed parameter, it is treated as the alias. If you invoke the program with two arguments,
the second of the two arguments is treated as the member name. Several examples are given below. Once you have all of the pieces in their respective locations, you can invoke the ED REXX program from any command line in TSO.

/* REXX EXEC */
parse upper arg FUNC NAME MBR
if NAME = "?" | NAME = "" then
  "ISPEXEC DISPLAY PANEL(EDPANEL)"
else
  do
    X = 1
    do until NAME = ALIAS | X > 34
      AL = "A"||X ; DS = "D"||X ; CN = "C"||X
      "ISPEXEC VGET ("AL DS CN") PROFILE"
      ALIAS = value('AL') ; ALIAS = value(ALIAS)
      X = X + 1
    end
    if X > 34 then
      do
        ZEDSMSG = "Invalid alias - "||NAME
        ZEDLMSG = "Alias does not exist, or you have a spelling error"
        "ISPEXEC SETMSG MSG(ISRZØØ1)"
        exit
      end
    else
      DS1 = value('DS') ; DSN = value(DS1)
      CT1 = value('CN') ; CNT = value(CT1)
      if CNT = "" then
        CNT = Ø
      else
        CNT = CNT + 1
        CNT_DATA = CT1 ; interpret CNT_DATA ' = CNT'
        "ISPEXEC VPUT ("CN") PROFILE"
      end
      code = LISTDSI(DSN)
      if code > Ø then
        do
          ZEDSMSG = "Invalid Dsn - "||NAME
          ZEDLMSG = "Dataset name associated with alias does not exist"
          "ISPEXEC SETMSG MSG(ISRZØØ1)"
          exit
        end
      if FUNC = "ED" then
        do
          if MBR = "" then
            "ISPEXEC EDIT DATASET("DSN")"
          else
            "ISPEXEC EDIT DATASET("DSN"("MBR")"
          if lastcc > Ø then "ISPEXEC SETMSG MSG(ISRZØØ2)"
        end
if FUNC = "BR" then
  do
    "ISPEXEC BROWSE DATASET("DSN")"
    if lastcc > Ø then "ISPEXEC SETMSG MSG(ISRZ002)"
  end
if FUNC = "SJ" then
  do
    ADDRESS "TSO"
    "SUBMIT "DSN"("MBR")"
    if lastcc > Ø then "ISPEXEC SETMSG MSG(ISRZ002)"
  end
end
exit

Here are some examples:

- ED – display EDPANEL
- BR ? – display EDPANEL
- ED JCL – edit the dataset associated with JCL
- BR SAS – browse the dataset associated with SAS
- SJ JCL BR14 – submit the BR14 member of the JCL dataset.

**Execute program with extended parameter**

**PROBLEM ADDRESSED**

The EXEC parameter has always been a useful and simple means of passing parameters to a program; rather than processing a file, five instructions suffice to access the EXEC parameter. In particular for compilers, there has been a dramatic increase in the number and range of parameters. However, since time immemorial (with regard to z/OS and its predecessors), the maximum length of the EXEC parameter that can be specified by JCL has remained at 100 characters. This may have been appropriate when (real) memory was literally worth its weight in gold, but nowadays this restriction is purely artificial. No robust program should have any problem in
handling an EXEC parameter of any specified length (the infamous buffer overflow problem with the associated security risks known from the Windows world underscores the problems associated with non-robust programs). Indeed, even some programs which explicitly state that they can process only parameter lists with a maximum of 100 characters, when invoked dynamically can actually process longer EXEC parameter lists (eg some COBOL compilers). A second problem concerns the somewhat abstruse rules for the continuation of a JCL EXEC PARM when commas, parentheses, and apostrophes are involved.

Some, but not all, compilers solve these problems by allowing the use of an options file that can contain additional compiler options.

EXTDPARM, the program described in this article, allows an extended EXEC parameter, specified in the SYSPARM file, to be passed to a program invoked dynamically. The name of the program to receive control is specified in the EXTDPARM EXEC parameter. This program is loaded from the current load library (STEPLIB, LOADLIB, etc). Thus, EXTDPARM can be used to invoke any program with an extended parameter provided the invoked program correctly processes its EXEC parameter.

SOLUTION

EXTDPARM dynamically invokes the program specified in the EXEC parameter with the parameter formed from the input specified by the SYSPARM file. The trailing blanks in each line of the SYSPARM file are removed. The resulting parameter list is formed by concatenating each line of the SYSPARM file; any special characters and leading blanks are passed unchanged.

Example 1:

```
ALPHA BETA
GAMMA
DELTA
```

produces the parameter list:

```
ALPHA BETA  GAMMA    DELTA
```
Example 2:

```
ALPHA BETA,
 'GAMMA '
DELTA
```

produces the parameter list:

```
ALPHA BETA, 'GAMMA ' DELTA
```

Invocation:

```
// EXEC PGM=EXTDPATM,PARM=pgmname
//STEPLIB DD DSN=loadlib,DISP=SHR
//SYSPARM DD *
parm1
parm2
...
```

DD statements are:

- **pgmname** – the name of the program to be invoked.
- **loadlib** – the name of the load library that contains EXTDPATM and pgmname. If required, additional load libraries can be concatenated.
- **parm1 ... parmn** – the parameters to be passed to pgmname.

Note: although SYSPARM is shown here assigned to the input stream, it can be assigned to any PS dataset that has RECFM=FB and LRECL<256 as file attributes.

Unless EXTDPATM detects a processing error – eg no program name specified (or name too long), or specified parameter too long (> 32760 characters, program constant) – it sets its completion code to that returned from the invoked program.

Error returns:

- **-1** – no external program name specified or name longer than eight characters.
- **-2** – parameter overflow.

**EXTDPATM**

```
TITLE 'Execute Program With Extended Parameter'
```
** EXTDPATM: Execute (load) program with extended parameter **

Invocation:
* // EXEC EXTDPATM,PARM=execname
* //SYSPARM DD * (RECFM=FB,LRECL<256)
*
* DD:SYSPARM contains the parameters to be passed to the
* specified program. Trailing blanks are removed from each
* line. The processed lines are concatenated together (any
* required delimiters must be specified explicitly, eg
* commas, leading blanks).
*
* Return:
* Return code from the executed program
* Error returns:
* -1: no external program name specified or name longer than
  8 characters.
* -2: parameter overflow
**

PRINT NOGEN
SPACE 1
EXTDPARM CSECT
EXTDPARM AMODE 31
EXTDPARM RMODE 24
* initialise addressing
STM   R14,R12,12(R13)             save registers
BASR  R12,Ø                  base register
USING *,R12
LA    R15,SA                 A(save-area)
ST    R13,4(R15)             backward ptr
ST    R15,8(R13)             forward ptr
LR    R13,R15                A(new save-area)
SPACE 1
LHI   R15,-1                 preload ReturnCode register
L     R2,Ø(R1)               pointer to parameters
SR    R1,R1                  zeroise R1
ICM   R1,3,Ø(R2)             length of program name
JZ    EXIT                   no exec name (=error)
CHI   R1,'EXECNAME          test length
JH    EXIT                   too long (=error)
BCTR  R1,Ø                   LengthCode(parm)
EX    R1,EXMOVE               store program name
SPACE 1
OPEN  (SYSPARM,(INPUT))      open SYSPARM file
LTR   R15,R15                test OPEN return code
JNZ   NOPARM                 open error -> file does not exist
LA    R2,EXECDATA
LR    R3,R2
AHI   R3,DATALEN             end of exec data
USING IHADCB,SYSPARM
READLOOP LHI R15,-2 preload ReturnCode
CR R2,R3 test for buffer overflow
JNL EXIT buffer overflow
GET SYSParm,(R2) read logical record
* remove trailing blanks
   LH R1,DCBLRECL logical record length
   AR R2,R1 address of record-end +1
TESTLOOP BCTR R2,Ø decrement address pointer
   CLI Ø(R2),C' ' test for blank
   JNE LOOPOUT non-blank found
   JCT R1,TESTLOOP test next character
* empty record
LOOPOUT LA R2,1(R2) correct address pointer
   J READLOOP read next record
FILEEOF DS ØH EOF(DD: SYSParm)
   S R2,=A(EXECdata) Length of buffer data
   STH R2,EXECLEN save length
   CLOSE (SYSParm)
SPACE 1
NOPARM DS ØH load and invoke program
   LINK EPLOC=EXECNAME,PARAM=(CALLParm),VL=1
SPACE 1
EXIT DS ØH job end
   L R13,4(R13) restore addr. of old save-area
   RETURN (14,12),RC=(15)
SPACE 1
EXMOVE MVC EXECNAME(Ø),2(R2)
SPACE 1
* symbolic register equates
RØ EQU Ø
R1 EQU 1
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R1Ø EQU 1Ø
R11 EQU 11
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15
TITLE 'Data Areas'
LTORG
SPACE 1
SYS Parm DCB DDNAME=SYS Parm,DS ORG=PS,MACRF=GM,EOD AD=FILEEOF
SPACE 1
SA       DS    18F  register save area
SPACE 1
EXECNAME DC    CL8' ' program name
SPACE
DATALEN  EQU   3276Ø
CALLParm DS    ØH
EXECLEN DC    H’Ø’
EXECData DS    CL(DATALEN)
DS    CL256 padding
SPACE 1
DCBD  DSORG=PS,DEV=DA
END

SAMPLE PROGRAM
IDENTIFICATION DIVISION.
PROGRAM-ID. COBPARM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
Ø1 EXECPARM.
Ø2 EXECPARM-LEN PIC 9(4) BINARY.
Ø2 EXECPARM-DATA PIC X(3276Ø).
PROCEDURE DIVISION USING EXECPARM.
    DISPLAY 'EXECPARM:' EXECPARM-DATA
    DISPLAY 'EXECPARM-LEN:' EXECPARM-LEN
STOP RUN.

SAMPLE INVOCATION
//        EXEC PGM=EXTDPATM,PARM=TESTPGM
//STEPLIB DD DSN=loadlib,DISP=SHR
//SYSPARM DD *
NOADATA  ADV
NOANALYZE  APOST
          ARITH(COMPAT) NOAWO
          BUFSIZE(8192)

OUTPUT
EXECPARM:NOADATA  ADV NOANALYZE  APOST ARITH(COMPAT) NOAWO
          BUFSIZE(8192)
EXECPARM-LEN:Ø065

Systems Programmer
(Germany) © Xephon 2005
System layout verification tool

For quite some time now, a typical mainframe installation environment comprises multiple CPUs with many partitions active.

The frequency with which the core software needs to be maintained imposes a precise operating system design that needs to clearly reflect the division between ‘target’, ‘distribution’, and ‘operational’ dataset types.

Generally speaking, a ‘target’ or ‘distribution’ dataset is one that has a ‘DDDEF’ definition in its respective SMP/E zone, whereas ‘operational’ datasets are all those that contain the customizations of a particular product or feature.

Moreover, the contents of the ‘target’ and ‘distribution’ datasets may change completely with every upgrade of the operating system, whereas the ‘operational’ datasets would have minimal updates if any, and so are maintained intact with each and every release of the core software.

This differentiation of datasets isn’t merely academic, but rather becomes almost obligatory if one intends to have an operating system and base software that must be easy to upgrade and maintain.

The fundamental characteristics of this software design are effectively summed up in the following technical documents:

- *OS/390 Maintenance Philosophy – An IBM View.*

Even the Serverpac dialogs push the systems programmer to dedicate the RESVOL volumes only to the ‘target’ datasets, while the SMP/E and ‘distribution’ datasets have separate volumes.

It is desirable that those ‘operational’ datasets that define the ‘image customization set’ get allocated on a different volume from the ones that go to a new ServerPac or CBPDO – in this case, one is often advised to take advantage of indirect cataloguing through the use of
the system symbols that are defined in the member of SYS1.PARMLIB(IEASYMxx).

Each one of us is full of good intentions, but in the end even in a job well done the odd error can slip through and in time create problems.

To this end I have written EXAMSYS, a batch utility that will examine (one volume at a time) all those volumes intended for the operating system.

The end result for each selected volume is a printout listing every file, clearly pointing out the relating SMP/E zone, the type of cataloguing used as well as the correct type to be used, and also the quantity of ‘operational’ datasets on that volume.

Having read through the report, one ultimately understands whether or not that particular volume has been correctly assigned and if any datasets have been allocated on the wrong logical volume.

To do all this I have used the standard IBM utilities as well as a few lines of code using the REXX language.

Everything can be easily modified, should one so desire, including the structure of the final printout (see Step/JCL with ICETOOL).

EXAMDSN REXX SAMPLE

/* REXX ------------------------------------ */
/* REXX RECORD MANIPULATION OF PRINTOUT FILE */
/* REXX ------------------------------------ */
TRACE OFF
" PROF NOPREF "
WK_FILEV = 'START'
WK_FILED = 'START'
WK_RCD = Ø
WK_RCV = Ø
"EXECIO Ø DISKR FILED (OPEN"
"EXECIO Ø DISKR FILEV (OPEN"
"EXECIO Ø DISKW FILEP (OPEN"
/* - EMPTY FILE -------------------------- */
IF WK_FILED = 'START' THEN DO
 "EXECIO 1 DISKR FILED "
 WK_RCD = RC
 IF WK_RCD > Ø THEN EXIT(98)
 PULL WK_FILED
 END
IF WK_FILEV = 'START' THEN DO
   "EXECIO 1 DISKR FILEV"
   WK_RCV = RC
   IF WK_RCV > Ø THEN EXIT(99)
   PULL WK_FILEV
END
/* - EMPTY FILE -------------------------- */

DO FOREVER

SELECT

WHEN WK_RCD > Ø
   THEN DO
      CALL PROC_PRINT
      "EXECIO 1 DISKR FILEV"
      WK_RCV = RC
      IF WK_RCV > Ø THEN LEAVE
      PULL WK_FILEV
   END

WHEN SUBSTR(WK_FILEV,Ø2,44) = SUBSTR(WK_FILED,3Ø,44)
   THEN DO
      CALL PROC_PRINT
      "EXECIO 1 DISKR FILED"
      WK_RCD = RC
      PULL WK_FILED
      "EXECIO 1 DISKR FILEV"
      WK_RCV = RC
      IF WK_RCV > Ø THEN LEAVE
      PULL WK_FILEV
   END

WHEN SUBSTR(WK_FILEV,Ø2,44) > SUBSTR(WK_FILED,3Ø,44)
   THEN DO
      "EXECIO 1 DISKR FILED"
      WK_RCD = RC
      PULL WK_FILED
   END

WHEN SUBSTR(WK_FILEV,Ø2,44) < SUBSTR(WK_FILED,3Ø,44)
   THEN DO
      CALL PROC_PRINT
      "EXECIO 1 DISKR FILEV"
      WK_RCV = RC
      IF WK_RCV > Ø THEN LEAVE
      PULL WK_FILEV
   END

OTHERWISE NOP
END
/* -------------------------------------- */
"EXECIO Ø DISKR FILEV (FINIS"
"EXECIO Ø DISKR FILED (FINIS"
"EXECIO Ø DISKW FILEP (FINIS"
EXIT(ØØ)
/* INTERNAL ROUTINE ---------------------- */
PROC_PRINT:
SELECT
WHEN INDEX(WK_FILEV,'.VTOCIX.') <> Ø THEN RETURN
WHEN INDEX(WK_FILEV,'.VVDS.')     <> Ø THEN RETURN
WHEN INDEX(WK_FILEV,'.DATA ')     <> Ø THEN RETURN
WHEN INDEX(WK_FILEV,'.INDEX ')    <> Ø THEN RETURN
WHEN INDEX(WK_FILEV,'.CATINDEX ') <> Ø THEN RETURN
OTHERWISE NOP
END
/* -------------------------------------- */
WK_DSNP = SUBSTR(WK_FILEV,Ø2,44)
WK_ZON  =  'OPERATIONAL    '
WK_DRC  =  ' NO  '
WK_CAT  =  ' NO  '
WK_FLG  =  '<<'
WK_DDN  =  '        '
/* -------------------------------------- */
LL = OUTTRAP(LINE.)
"LISTC ENT("WK_DSNP") VOL"
IF RC = Ø THEN DO
  WK_CAT = ' YES '
  I = 4
  DO UNTIL I = 9
    IF INDEX(LINE.I,"DEVTYPE------X'ØØØØØØØØ'") > Ø
      THEN WK_DRC = ' YES '
    I = I + 1
  END
END
LL = OUTTRAP(OFF)
/* -------------------------------------- */
SELECT
WHEN  WK_RCD > Ø THEN NOP
WHEN  SUBSTR(WK_FILEV,Ø2,44) = SUBSTR(WK_FILED,3Ø,44)
  THEN DO
    WK_ZON =  SUBSTR(WK_FILED,1Ø1,15)
    WK_DDN =  SUBSTR(WK_FILED,Ø2,Ø8)
EXAMOUT REXX SAMPLE

/* REXX ---------------------------------- */
/* REXX RECORD MANIPULATION OF OUTPUT FILE */
/* REXX ---------------------------------- */
TRACE OFF
WK_EMPTY = 'NOK'
WK_ZONE = '
"EXECIO Ø DISKR INPØØØ (OPEN"
"EXECIO Ø DISKW SORTIN (OPEN"
/* -------------------------------------- */
READØØ:
DO FOREVER
"EXECIO 1 DISKR INPØØØ "
WK_RC = RC
IF WK_RC > Ø THEN LEAVE
PULL MY_INPØØØ
IF INDEX(MY_INPØØØ,'DDDEF ENTRIES') = Ø
THEN DO
WK_SORTIN = SUBSTR(MY_INPØØØ,01,100)WK_ZONE
PUSH WK_SORTIN
"EXECIO 1 DISKW SORTIN "
END ELSE DO
WK_EMPTY = 'OK'
WK_ZONE = SUBSTR(MY_INPØØØ,02,07)
END
ENDØØ:
"EXECIO Ø DISKR INPØØØ (FINIS"
"EXECIO Ø DISKW SORTIN (FINIS"
/* -------------------------------------- */
IF WK_EMPTY = 'NOK' THEN EXIT(20)
/* -------------------------------------- */
IF WK_RC <> 2 THEN EXIT(50)
/* -------------------------------------- */
ADDRESS "LINKMVS" "ICEMAN"
IF RC <> Ø THEN EXIT(40)
/* -------------------------------------- */
"EXECIO Ø DISKW WORKEND (OPEN"
"EXECIO Ø DISKR SORTOUT (OPEN"
WK_REC1 = '$$START.$$'
WK_REC2 = '

DO FOREVER
  SELECT
  WHEN WK_REC1 = '$$END.$$'
    THEN LEAVE
  WHEN WK_REC1 = '$$START.$$'
    THEN CALL READ_ALL
  WHEN SUBSTR(WK_REC1,2,44) = SUBSTR(WK_REC2,2,44)
    THEN DO
      WK_ZON1 = SUBSTR(WK_REC1,1,1,7)
      WK_ZON2 = SUBSTR(WK_REC2,1,1,7)
      WK_WORKEND = SUBSTR(WK_REC1,01,099) WK_ZON1 WK_ZON2
      PUSH WK_WORKEND
      "EXECIO 1 DISKW WORKEND"
      CALL READ_ALL
      END
  WHEN SUBSTR(WK_REC1,2,44) <> SUBSTR(WK_REC2,2,44)
    THEN DO
      WK_WORKEND = WK_REC1
      PUSH WK_WORKEND
      "EXECIO 1 DISKW WORKEND"
      WK_REC1 = WK_REC2
      CALL READ_ONE
      END
  OTHERWISE NOP
  END
END
"EXECIO Ø DISKR SORTOUT (FINIS"
"EXECIO Ø DISKW WORKEND (FINIS"
EXIT(ØØ)
/* INTERNAL ROUTINE ---------------------- */
READ_ALL:
"EXECIO 1 DISKR SORTOUT"
  IF RC > Ø THEN WK_REC1 = '$$END.$$'

ELSE PULL WK_REC1

READ_ONE:
IF WK_REC1 = '$$.END.$$' THEN RETURN

"EXECIO 1 DISKR SORTOUT"
IF RC > Ø THEN WK_REC2 = '$$.END.$$'
ELSE PULL WK_REC2

RETURN

SAMPLE JCL TO RUN EXAMSYS

//...... JOB ..............,CLASS=.,MSGCLASS=.,REGION=ØM,COND=(Ø,GT)
//** --------------------------------------------------------- **
//** BEFORE SUBMITTING THE JOB:                              **
//** > IN ST1ØØ INDICATE:                                    **
//** THE CORRECT CSI NAME ON THE 'SMPCI' DD CARD;             **
//** THE CORRECT ZONE NAMES ON THE 'SMPCNTL' DD CARD.         **
//** > IN ST4ØØ INDICATE:                                    **
//** THE NAME OF THE VOLUME TO BE EXAMINED IN 'SYSIN' CARD.**
//** THE CORRECT VOL=SER ON THE 'DISKØØ' DD CARD.          **
//** --------------------------------------------------------- **
//ST1ØØ EXEC PGM=GIMSMP
//SMPCI DD DISP=SHR,DSN='your_.GLOBAL.CSI'
//SMPLIST DD DSN=&SMPLIST,DISP=(MOD,PASS),UNIT=VIO,
//  SPACE=(CYL,3),DCB=(LRECL=121,BLKSIZE=726Ø,RECFM=FBA)
//SYSOUT DD SYSOUT=*  
//SMPLOG DD DUMMY
//SMPLOGA DD DUMMY
//SMPRPT DD DUMMY
//SMPOUT DD DUMMY
//SMPCNTL DD *
// SET BOUNDARY(GLOBAL).
// LIST DDDEF.  
// SET BOUNDARY(your_target_zone).
// LIST DDDEF.
// SET BOUNDARY(your_distribution_zone).
// LIST DDDEF.
/*
//** --------------------------------------------------------- **
//ST2ØØ EXEC PGM=ICEMAN
//SORTIN DD DSN=&SMPLIST,DISP=(OLD,PASS)
//SORTOUT DD DSN=&DDDEF1,DISP=(,PASS),UNIT=VIO,
//  SPACE=(CYL,3),DCB=*.ST1ØØ.SMPLIST
//SYSOUT DD DUMMY
//SYSIN DD *
// SORT FIELDS=COPY
// INCLUDE COND=(12,17,CH,EQ,C'DATASET         =',
/*
OR,10,13,CH,EQ,C'DDDEF ENTRIES')
/*
/** ** EXEC REXX - EXAMOUT ------------------------------- **
//ST300 EXEC PGMD=IKJEFT01,PARM='%EXAMOUT'
//INP000 DD DSN=&&DDDEF1,DISP=(OLD,PASS)
//SORTIN DD DSN=&&DDDEF2,DISP=,(PASS),UNIT=VIO,
// SPACE=(CYL,3),DCB=* .ST100.SMPLIST
//SORTOUT DD DSN=&&DDDEF3,DISP=(,PASS),UNIT=VIO,
// SPACE=(CYL,3),DCB=* .ST100.SMPLIST
//WORKEND DD DSN=&&DDDEND,DISP=(,PASS),UNIT=VIO,
// SPACE=(CYL,3),DCB=* .ST100.SMPLIST
//SYSPROC DD DISP=SHR,DSN=your_sysproc
//SYSPRINT DD SYSOUT=* 
//SYSOUT DD DUMMY
//SYSIN DD *
SORT FIELDS=(30,44,CH,A) 
/ */
/** ** EXEC REXX - EXAMDSN ------------------------------- **
//ST600 EXEC PGMD=IKJEFT01,PARM='%EXAMDSN'
//FILEV DD DSN=&&VTOCP,DISP=(OLD,PASS)
//FILED DD DSN=&&DDDEND,DISP=(OLD,PASS)
//FILEP DD DSN=&&OUTP0,DISP=(,PASS),UNIT=VIO,
// SPACE=(CYL,3),DCB=(LRECL=90,BLKSIZE=6030,RECFM=FB)
//SYSPRINT DD DUMMY

//SYSTSPRT DD DUMMY
//SYSPROC DD DISP=SHR,DSN=*,.ST300.SYSPROC
//SYSTSIN DD DUMMY
/** -------------------------------------------------------- **
//ST700 EXEC PGM=ICETOOL
//TOOLMSG DD DUMMY
//DFSMSG DD DUMMY
//IN1 DD DISP=(OLD,PASS),DSN=&&OUTP0
//REPORT DD SYSOUT=* 
//TOOLIN DD *
DISPLAY FROM(IN1) LIST(REPORT) DATE TIME PAGE 
  TITLE(' SYSTEM DESIGN CHECK ') -
  HEADER('DATASET NAME ') ON(01,44,CH) -
  HEADER(' SMP/E DDDEF ') ON(45,08,CH) -
  HEADER(' SMP/E ZONES ') ON(54,16,CH) -
  HEADER('INDIRECT/CAT ') ON(70,05,CH) -
  HEADER('CATALOGUED ') ON(76,05,CH) -
BLANK LINES(63)
/*
PRINTOUT RESULT
01/28/03 13:39:23 - 1 - SYSTEM DESIGN CHECK

<table>
<thead>
<tr>
<th>DATASET NAME</th>
<th>SMP/E DDDEF</th>
<th>SMP/E ZONES</th>
<th>INDIRECT/CAT</th>
<th>CATALOGUED</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOP.SAOPEXEC</td>
<td>SAOPEXE</td>
<td>TARGET_zone</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>AOP.SAOPMENU</td>
<td>SAOPMEN</td>
<td>TARGET_zone</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>AOP.SAOPPENU</td>
<td>SAOPPEN</td>
<td>TARGET_zone</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>GLD.SGLDLNK.COPY</td>
<td>OPERATIONAL</td>
<td>NO</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>GSK.SGSKLOAD</td>
<td>SGSKLOA</td>
<td>TARGET_zone</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>ICA.SICALMOD</td>
<td>SICALMO</td>
<td>TARGET_zone</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>IOE.SIOEPROC</td>
<td>SIOEPRO</td>
<td>TARGET_zone</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>ISF.SISFEXEC</td>
<td>SISFEXE</td>
<td>TARGET_zone</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>ISF.SISFLINK</td>
<td>SISFLIN</td>
<td>TARGET_zone</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>ISF.SISFLOAD</td>
<td>SISFLOA</td>
<td>TARGET_zone</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>SYS1.ADGTPLIB</td>
<td>ADGTPLI</td>
<td>DISTRIBUTION_zone</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>SYS1.AERBPSV</td>
<td>AERBPSW</td>
<td>DISTRIBUTION_zone</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>SYS1.PARMLIB</td>
<td>OPERATIONAL</td>
<td>NO</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>SYS1.VTAMLST</td>
<td>OPERATIONAL</td>
<td>YES</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>TCPIP.SEZACMTX</td>
<td>SEZACMT</td>
<td>TARGET_zone</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Massimo Ambrosini (Italy) © Xephon 2005
Comparing two files

The following program was written to check whether two files have identical contents. The files being compared can be KSDS, RRDS, ESDS, or non-VSAM. The files have DDnames INFILE1 and INFILE2. Each record is compared with the same record of the other file. If one of the records is shorter than the other, the shorter record is padded with the character indicated by variable PADCHAR within the program (space by default), and the rules of the CLCL (compare logical long) instruction apply. This is useful, for example, if you want to compare an ordinary 80-byte sequential file, but where only the first 50 bytes are meaningful and the remaining are spaces, with a 50-byte VSAM file.

If an unequal record is found, the program sends out a message to SYSPRINT, indicating that record number, and continues the comparison. If a predefined number of unequal records is attained (as defined by variable DIFLIMIT), the program terminates. Otherwise, the program continues until it reaches the end of both files. If one of the files ends first, the program continues to read the other up to its end, but without performing any more comparisons, in order to find out how many records each file has. In the end, it sends a message with that information.

You can also compare just part of each record. For example, you have code in the first file which is 10 bytes long and occurs at offset 23, and you want to know if it matches the second file, where that code occurs at offset 175. For this, pass a parameter to the program, with two pairs of length-offset values, with each value separated by commas. The first pair relates to INFILE1 and the second pair to INFILE2:

```plaintext
//STEP1  EXEC PGM=VCOMPARE,PARM='10,23,10,175'
//INFILE1  DD DISP=SHR,DSN=first.file
//INFILE2  DD DISP=SHR,DSN=second.file
//SYSPRINT DD SYSOUT=*    
```

This way, only the specified bytes are compared, and the rest of each record is ignored.
If no parameter is given, the entire records are compared, as initially explained. Parameters are positional, and can be partially omitted. For example, if you just want to compare the first 25 bytes of each record (that means, with the default offset zero), you can code either of these:

```
//EXEC PGM=VCOMPARE,PARM='25,Ø,25,Ø'
//EXEC PGM=VCOMPARE,PARM='25,,25'
```

This also is applicable if, as in the first example, you just want to compare the first 50 bytes of the sequential file against the entire 50-byte VSAM. Since this last value is the default, because it is the VSAM record length, you can omit it, and just specify the length for the first file:

```
//EXEC PGM=VCOMPARE,PARM='5Ø'
```

which, in this particular case, would be identical to:

```
//EXEC PGM=VCOMPARE,PARM='50,Ø,50,Ø'
//EXEC PGM=VCOMPARE,PARM='50,,50'
```

This way, the remaining 30 bytes of the seqential file would be ignored, and only the first 50 bytes would be considered.

VCOMPARE SOURCE CODE

```"`
*====================================================================*
*                                                                    *
* VCOMPARE - Compare two files. The files can be VSAM, sequential,    *
* or both, with fixed or variable length.                            *
* Files are assigned to DDnames INFILE1 and INFILE2.                 *
* Records are compared in a parallel fashion. If two records do not  *
* have the same length, the smaller is considered to be padded with  *
* the character stored in variable PADCHAR (space by default).       *
* If inequal records are found, the program prints a message with    *
* the record number and increments a counter. If a certain number of *
* unequal records is attained, the program terminates.               *
* That number is set in variable DIFLIMIT.                           *
*                                                                    *
* The program also states how many records each file has. If a file   *
* ends sooner than the other, the program continues to read the      *
* longer file until it ends, to determine the number of records, but *
* no more comparisons take place.                                    *
*                                                                    *
* Records can be compared totally or in part. For partial comparison*
* specify a length-offset pair for each file, with each value        *
```
* separated by commas. Values are positional. If a value is not * 
* specified, the defaults are assumed (the full length of each * 
* record from offset zero). The first pair of values concerns file1 * 
* and the second pair concerns file2.                                *
*====================================================================*
&PROGRAM SETC 'VCOMPARE'
&PROGRAM AMODE 31
&PROGRAM RMODE 24
&PROGRAM CSECT
SAVE (14,12)
LR R12,R15
USING &PROGRAM,R12
ST R13,SAVEA+4
LA R11,SAVEA
ST R11,8(R13)
LR R13,R11
B GETPARMS
DC CL16' &PROGRAM 2.1'
DC CL8'&SYSDATE'
*====================================================================*
* Separate input parameter into its components, convert them to 
* binary form, and store them in fields PARM1 thru PARM4. If any of 
* the parms does not exist, those fields will remain with low-values. 
* For each parm specified, set the corresponding flag field to 1, in 
* order to allow CLI comparisons later on.                          *
*====================================================================*
* GETPARMS DS ØH
LR R2,R1               Copy parameter pointer to R2.
L R2,Ø(Ø,R2)          Load parm address
LH R3,Ø(R2)            Load parm length in R3
LTR R3,R3               Any parm entered?
BZ OPENPRT              No
* 
LR R6,R2                  End of all parms?
AR R6,R3                 R6: point after end of parms
LA R6,2(Ø,R6)           Skip 2 bytes of parmlength
LA R2,2(Ø,R2)           Skip 2 bytes of parmlength
LR R4,R2                 R4: Current char to check
LA R11,PARM1             Area to keep parms in binary form
XR R9,R9                 Clear length counter
* 
LOOPARMS EQU             *
CR R4,R6                 End of all parms?
BNL CONVERT              Yes, go convert the last one
CLI Ø(R4),',,'          Comma found?
BE CONVERT              Yes, go convert parm
LA R9,1(Ø,R9)           Increment index (char counter)
LA R4,1(Ø,R4)          Increment pointer (current char)
B   LOOPARMS            And continue
*
CONVERT EQU  *
LTR R9,R9                Any chars in current parm?
BZ CONVERT2              No, skip pack and cvb instructions
S R9,=F'1'                Sub one for ex
EX R9,PACKEX             Execute pack
LA R9,1(Ø,R9)           Increment again
CVB R7,PARMPACK         Convert to binary into R7
ST R7,Ø(R11)            And store it in R11 (Parm1 to 4)
*
CONVERT2 EQU  *
CR R4,R6                 End of all parms?
BNL SETFLAG1             Yes, move ahead
AR R2,R9                 Add length to base pointer
LA R2,1(Ø,R2)            And skip comma
XR R9,R9                 Reset length
LR R4,R2                 R4: Current char
LA R11,4(Ø,R11)          Point next binary parm storarea
B   LOOPARMS
*
SETFLAG1 CLC  PARM1,=F'Ø'        Parm1 specified?
BE SETFLAG2             No, try next
MVI P1FLAG,C'1'         Yes, set flag to 1
SETFLAG2 CLC  PARM2,=F'Ø'        Same for others
BE SETFLAG3
MVI P2FLAG,C'1'
SETFLAG3 CLC  PARM3,=F'Ø'
BE SETFLAG4
MVI P3FLAG,C'1'
SETFLAG4 CLC  PARM4,=F'Ø'
BE OPENPRT
MVI P4FLAG,C'1'
*
*====================================================================*
* Check what kind of files we have and try to open them.  *
* First attempt is to open as VSAM. If Error, assume non-VSAM file. *
* If VSAM, test ACB for ESDS. If ESDS, modify RPL accordingly. *
*====================================================================*
*
OPENPRT DS  ØF               Open sysprint
OPEN (SYSPRINT,OUTPUT)      for displaying messages
*
OPENACB1 EQU  *
OPEN INFILEA1              Open ACB for VSAM file
LTR R15,R15                If error, go open DCB for seq file
BNZ OPENDCB1
TESTCB ACB=INFILEA1,       X
ATRB=ESDS                 Check if VSAM ESDS
BNE OPENACB2  No, go open second file

* ESDSFIL1 EQU *  X
MODCB RPL=INFILER1, Modify RPL for ESDS
OPTCD=ADR
B OPENACB2

* OPENDCB1 EQU *  Open DCB (sequential file)
OPEN (INFILED1,INPUT)
LTR R15,R15
BNZ ERRMSG2

* MVI FILETYP1,C'S'  Set flag sequential type (nonVSAM)
LA R2,INFILED1  Address IHADCB of input file1
USING IHADCB,R2
TM DCBRECFM,DCBBIT1  Is recfm V or U (B'xlxxxxxx)
BNO OPENACB2  No, jump ahead
MVI FILEVAR1,C'U'  set recfm undefined
TM DCBRECFM,DCBBITØ  Is recfm U (B'11xxxxxx)
BO OPENACB2  Yes, jump ahead
MVI FILEVAR1,C'V'  set recfm variable

* * OPENACB2 EQU *  Open ACB for VSAM file
OPEN INFILEA2  If error, go open DCB for seq file
LTR R15,R15
BNZ OPENDCB2
TESTCB ACB=INFILEA2,  X
ATRB=ESDS  Check if VSAM ESDS
BNE READFILS  No, jump ahead

* ESDSFIL2 EQU *  X
MODCB RPL=INFILER2, Modify RPL for ESDS
OPTCD=ADR
B READFILS

* OPENDCB2 EQU *  Open DCB (sequential file)
OPEN (INFILED2,INPUT)
LTR R15,R15
BNZ ERRMSG2

* MVI FILETYP2,C'S'  Set flag sequential type (nonVSAM)
LA R11,INFILED2  Address IHADCB of input file2
DROP R2
USING IHADCB,R11
TM DCBRECFM,DCBBIT1  Is recfm V or U (B'xlxxxxxx)
BNO READFILS  No, jump ahead
MVI FILEVAR2,C'U'  set recfm undefined
TM DCBRECFM,DCBBITØ  Is recfm U (B'11xxxxxx)
BO READFILS  Yes, jump ahead
MVI FILEVAR2,C'V' set recfm variable
*
*====================================================================*
* Now enter a loop where we read a pair of records and compare them
* If they are equal, continue reading another pair.
* If they are different, print a message with the record number within
* the file and continue. If the maximum limit of different records
* is attained, terminate the program.
*====================================================================*
*
READFILS EQU *
  XR R8,R8 Record count for file1
  XR R9,R9 Record count for file2
READ1 EQU *
  CLI ENDF1,C'F' End of file1 already happened?
  BE READ2 Yes, just read file2
  LA R8,1(Ø,R8) Increment file1 record counter
  CLI FILETYP1,C'V'
  BNE READSEQ1
*
READVSA1 EQU *
  GET RPL=INFILER1 Read VSAM file
  LTR R15,R15 End of file?
  BNZ ENDFILE1
  L R4,VAREA1 Get address of data in R4.
  SHOWCB RPL=INFILER1, AREA=LRECL1, LENGTH=4, FIELDS=RECLEN
  L R5,LRECL1 Get record length in R5
  B READ2
*
READSEQ1 EQU *
  DROP R11
  USING IHADC8,R2
  GET INFILED1 Read sequential (locate method)
  LR R4,R1 copy address of data to R4.
  LH R5,DCBLRECL Load R5 with record length.
  CLI FILEVAR1,C'V' Is recfm variable?
  BNZ READ2 No, jump ahead.
  LA R4,4(Ø,R4) Yes, skip 4 bytes of RDW
  SH R5,=H'4' And reduce record length.
*
READ2 EQU *
  CLI ENDF2,C'F' End of file2 already happened?
  BE READ1 Yes, just read file1
  LA R9,1(Ø,R9) Increment file2 record counter
  CLI FILETYP2,C'V'
  BNE READSEQ2*
READVS2 EQU *
GET RPL=INFILER2 Read VSAM file
LTR R15,R15 End of file?
BNZ ENDFILE2
L R6,VAREA2 Get address of data in R6
SHOWCB RPL=INFILER2,  
  AREA=LRECL2,  
  LENGTH=4,  
  FIELDS=RECLEN
L R7,LRECL2 Get record length in R7
B COMPARE

READSEQ2 EQU *
DROP R2
USING IHADCB,R11
GET INFILED2 Read sequential (locate method)
LR R6,R1 copy address of data to R6.
LH R7,DCBLRECL Load R7 with record length.
CLI FILEVAR2,C'V' Is recfm variable?
BNZ COMPARE No, jump ahead.
LA R6,4(Ø,R6) Yes, skip 4 bytes of RDW
SH R7,H'4' And reduce record length.

COMPARE EQU *
CLI ENDF1,C'F' If any of the files already ended,
BE READ2 no comparison is necessary.
CLI ENDF2,C'F' Just continue to read the other
BE READ1 until it ends also.

ASKFLAG1 CLI P1FLAG,C'Ø' If parm1 (length) not zero,
BE ASKFLAG2 assume parm1 length for file1
L R5,PARM1
ASKFLAG2 CLI P2FLAG,C'Ø' If parm2 (offset) not zero,
BE ASKFLAG3 add it to the record pointer
A R4,PARM2
ASKFLAG3 CLI P3FLAG,C'Ø' Same for file 2 parms.
BE ASKFLAG4
L R7,PARM3
ASKFLAG4 CLI P4FLAG,C'Ø'
BE ASKNOMOR
A R6,PARM4
ASKNOMOR ICM R7,B'1000',PADCHAR Insert padchar in R7

COMPLOOP EQU *
CLCL R4,R6 Compare strings
BZ READ1 Strings are equal
BL DIFERENT Strings are different
BH DIFERENT Strings are different
B COMPLOOP Equal so far, continue

DIFERENT EQU *
XR  R0,R0
AH  R0,DIFCOUNT  Increment different record counter
AH  R0,=H'1'
STH R0,DIFCOUNT
CH  R8,DIFLIMIT  Different limit attained?
BE  EXIT2       Yes, exit
LR  R0,R8       Prepare different recnum
BAL R10,UNPACK  for display
MVC DIFNUM,OUT10
PUT SYSPRINT,DIFMSG  Send message
B  READ1       And continue with next record

*   EQU *
S   R8,=F'1'
LR  R0,R8       Prepare number of records
BAL R10,UNPACK  for display
MVC ENDM1,OUT10
PUT SYSPRINT,ENDMSG1  Send message
MVI ENDF1,C'F'
CLI ENDF2,C'F'
BE  EXIT0
B  READ2

*   EQU *
S   R9,=F'1'
LR  R0,R9
BAL R10,UNPACK  for display
MVC ENDM2,OUT10
PUT SYSPRINT,ENDMSG2
MVI ENDF2,C'F'
CLI ENDF1,C'F'
BE  EXIT0
B  READ1

*====================================================================*
*         Close files and exit                                      *
*====================================================================*

EXITØ  EQU *
CLC  DIFCOUNT,=H'0'
BNE  EXIT1
PUT  SYSPRINT,NODIFMSG
B  EXIT1

EXIT2  EQU *
PUT  SYSPRINT,LIMITMSG

EXIT1  EQU *
CLOSE INFILE1
CLOSE INFILE2
CLOSE INFILEA1
CLOSE INFILEA2
CLOSE SYSPRINT
L R13,SAVEA+4
LM R14,R12,12(R13)
XR R15,R15
BR R14
*
*====================================================================*
* Subroutines and work areas
*====================================================================*
*
UNPACK EQU * Binary to display:
CVD RØ,REGDECIM Convert binary to packed decimal
UNPK OUT12,REGDECIM and unpack
BR R1Ø Return
*
ERRMSG1 EQU *
PUT SYSPRINT,=CL8Ø'>>> Error opening input file INFILE1'
B EXIT1
ERRMSG2 EQU *
PUT SYSPRINT,=CL8Ø'>>> Error opening input file INFILE2'
B EXIT1
*
INFILEA1 ACB DDNAME=INFILE1 VSAM ACB
INFILER1 RPL ACB=INFILEA1, VSAM RPL X
OPTCD=LOC, Locate method X
AREA=VAREA1, Record buffer address X
ARG=CHAVE1 Only needed for rrds
*
INFILED1 DCB DSORG=PS,MACRF=(GL), For sequential files X
EODAD=ENDFILE1, X
DDNAME=INFILE1
*
INFILEA2 ACB DDNAME=INFILE2 VSAM ACB
INFILER2 RPL ACB=INFILEA2, VSAM RPL X
OPTCD=LOC, Locate method X
AREA=VAREA2, Record buffer address X
ARG=CHAVE2 Only needed for rrds
*
INFILED2 DCB DSORG=PS,MACRF=(GL), For sequential files X
EODAD=ENDFILE2, X
DDNAME=INFILE2
*
SYSPRINT DCB DSORG=PS,MACRF=(PM), X
LRECL=Ø, X
DDNAME=SYSPRINT
*
LTORG
SAVEA DS 18F
VAREA1 DS F Address of record buffer (VSAM)
CHAVE1 DS F Record key (rrds - VSAM)
LRECL1 DS F Record length (VSAM)
VAREA2 DS F Address of record buffer (VSAM)
CHAVE2 DS F Record key (rrds - VSAM)
LRECL2 DS F Record length (VSAM)
FILETY1 DC C'V' Flag preset for VSAM
FILETY2 DC C'V' Flag preset for VSAM
FILEVAR1 DC C'F' Flag preset to recfm F (if nonVSAM)
FILEVAR2 DC C'F' Flag preset to recfm F (if nonVSAM)
ENDF1 DC C' ' Flag for end of file 1
ENDF2 DC C' ' Flag for end of file 2
PADCHAR DC C' ' Fill char for different reclen
DIFLIMIT DC H'2Ø' Max different records
DIFCOUNT DC H'Ø' Different record count area

PACKEX PACK PARMPACK,Ø(Ø,R2) Pack from input parm to parmpack
PARMPACK DS D Pack and convert to binary areas
PARM1 DC F'Ø' for input parameters
PARM2 DC F'Ø'
PARM3 DC F'Ø'
PARM4 DC F'Ø'
P1FLAG DC C'Ø' Flags are set to 1
P2FLAG DC C'Ø' if parms 1 to 4 have a value
P3FLAG DC C'Ø' other than the initial zero.
P4FLAG DC C'Ø'

DS ØD Convert to decimal and unpack
REGDECIM DS CL9 areas for output numbers
DS ØF
OUT12 DS ØCL12
OUT1Ø DS CL1Ø
DS CL2

NODIFMSG DC CL8Ø'** No differences found in compared records **'
LIMITMSG DC CL8Ø'++ Max number of different records attained ++'
DIFMSG DC C'Differences found in record number :'
DIFNUM DS CL1Ø
DC CL4Ø' '
ENDMSG1 DC C'Number of records of INFILE1 . . . :
ENDNUM1 DS CL1Ø
DC CL4Ø' '
ENDMSG2 DC C'Number of records of INFILE2 . . . :
ENDNUM2 DS CL1Ø
DC CL4Ø' '

DCBD DSORG=PS Ihadcb map (addressed by R2 for
YREGS file1 and by R11 for file2)
END

Systems Programmer
(Portugal) © Xephon 2005
INTRODUCTION

As is commonly known, beginning with z/OS V1R3, compatibility-mode is no longer available and an IPLed system will run in WLM goal-mode only. This means that each installation will be required to have a service definition installed and a WLM policy activated. Once a service definition is in place and the system is running in goal-mode, performance analysts are faced with the task of trying to understand what is going on in the system.

On the other hand, it may happen that, even though an installation is running in goal-mode for quite some time and everything is performing quite well, there are still changes, such as workload, software, or hardware changes, that should cause one to review, re-evaluate, and perhaps to modify WLM goals. This is where using a performance reporter product can be useful.

One of the best ways to review the WLM performance metrics is through the use of both real-time monitors (RMF Monitor III SYSSUM report, for example) and a postprocessor.

There are several areas one will want to look at to quickly gain knowledge of how a given workload is performing in relation to the goals that have been set in the service policy. One should keep in mind the fact that WLM uses three primary metrics to define how it should manage workloads – importance levels, service objectives, and performance index (PI):

- Importance level identifies the service classes according to the order in which WLM is to try to satisfy stated objectives, i.e., the order they should receive/donate resources. Since WLM dynamically adjusts the resources, the importance level determines how those adjustments are to be made and in what order. It was noticed that there is a strong temptation to place too many units of work into the upper importance levels and thus to overload WLM’s decision-making capability. The consequence
of this is that a high importance-level workload that fails to meet its objectives will invariably prevent lower importance level work from being examined.

- The defined service objectives categories (response time, response time percentile, velocity, discretionary) are telling WLM what the standard of measure will be.

- Performance index (PI) is used to evaluate how well the stated objectives are being met. WLM uses performance index in conjunction with importance level to determine what action (if any) should be taken. Most people are aware that a PI greater than 1 indicates that goals are not being met, while a PI of less than 1 indicates they are being exceeded.

Once the work has been classified into service classes with defined goals, the performance index tells us whether the workload on the system is meeting its WLM policy-defined goals, or that nothing is acting in the way we thought it would. Besides the obvious point that a service class is missing its objectives, this may also indicate that a particular objective is simply too aggressive for WLM to ever satisfy. What if you have noticed that a service class isn’t performing well because of a WLM-managed resource such as CPU, or that a Sysplex performance index is significantly less than a local performance index? For those performance analysts who have a grasp of Workload Manager concepts, constraints, and analysis techniques, the WLM postprocessor can significantly reduce the time required to perform daily analysis of system performance.

COLLECTING WLM DATA

As mentioned above, Workload Manager periodically assesses the performance of each service class period by comparing the performance achieved by the service class period against the performance goals defined for the service class period. WLM does this by sampling the state of the service class four times per second. This assessment is done at each goal importance level. In this way, WLM can determine whether the service class is using resources or whether the service class is being delayed in a manner that may be
adjustable. These sorts of delay, over which WLM can exert no control, are discarded in this assessment and do not contribute directly to WLM’s decision making. Idle time periods are excluded from the samples collection.

In order to document its decisions, WLM creates several SMF records (type 99) for each policy interval, or approximately once every 10 seconds. They can be useful in analysing and understanding the performance characteristics of a site’s workload. The records contain performance data for each service class period, a trace of SRM actions, the data SRM used to decide which actions to take, and the internal controls SRM uses to manage work. This can help the performance analyst to determine in detail what SRM is doing to meet workload goals defined with respect to other work, and the types of delays the work is experiencing.

Before proceeding any further it might be helpful to clarify the difference between SMF record type 72 (RMF Workload Activity) subtype 3 and record type 99 (System Resource Manager Decisions) subtype 6 since these two do overlap in some of their content. However, they have two significant differences.

Record type 99(6) contains local and Sysplex-level performance index values as calculated at policy adjustment time (in fact, this subtype contains no new data – everything in it is already in other subtypes of the type 99, but the new record compacts the needed data in one subtype so that one can afford to write that subtype 6 record and can suppress all other subtypes to reduce data volume).

The record type 72(3) does not include a PI value but does contain all the data needed to calculate an average PI for the RMF recording interval. Furthermore, type 99(6) provides the data on WLM’s internally-used dynamic service classes, but it is only type 72(3) that contains resource consumption data.

A detailed description of the layout of SMF type 99 record and its subtypes can be obtained from the MVS System Management Facilities (SMF) - SA22-7630-03 manual.

For information about how to use type 99, see z/OS MVS Programming: Workload Management Services.
For information about workload management, see z/OS MVS Planning: Workload Management.

The mapping macro, IRASMF99, for this record is supplied in SYS1.AMODGEN.

Because SMF type 99 records are written approximately every 10 seconds, one should write them only for certain time periods and define them (in SMFPRMxx member) like:

```
SYS(NOTYPE(99))
SYS(TYPE(99(6))
```

**CODE**

In order to provide a starting point from which one can begin to gather information about the system, an example of the WLM postprocessor JCL statements is included below.

The code is a three-part stream. In the first part (COPY996) selected SMF records (selection being defined by INCLUDE’s condition) are copied from the SMF dataset to a VB file, which can be used as a base of archived records. In the second part (WLM99), the captured records are being formatted by invoking REXX EXEC (WLMPP). In the last part (RPT996), the formatted records are being read and a report produced. The field reformatting capability of DFSORT’s ICETOOL was used to produce a report from the WLMPP output. For each service class, related information is produced – class name, period, local and Sysplex performance index, goal type defined and goal value measured, period importance, goal percentile, dispatching and I/O priority.

This job stream can be used to create a flexible report of those metrics that can quickly provide us, at a glance, with data about service classes which are and are not meeting specified goals, local and Sysplex-level PI. From a WLM perspective, a daily or weekly review of the reports should be used to provide a set of measurements to track and provide information for trend analysis. One should choose a busy time frame (1-3 hours) to use as a measurement period for this purpose.

//COPY996 EXEC PGM=ICETOOL
// TOOLMSG DD SYSOUT=*  
// DFSMSG DD SYSOUT=*  
// RAWSMF DD DSN=hlq.SMFdumpw,DISP=SHR  
// SMF99  DD DSN=your.copied.by.sort.to.VB.smf.dataset,  
//         SPACE=(CYL,(1)),UNIT=SYSDA,  
//         DISP=(NEW,PASS),  
//         DCB=(RECFM=VB,LRECL=32756,BLKSIZE=32760)  
// TOOLIN DD *  
COPY FROM(RAWSMF) TO(SMF99) USING(SMFI)  
// SMFICNTL DD *  
OPTION SPANINC=RC4,VLSHRT  
INCLUDE COND=(6,1,BI,EQ,99,AND,23,2,BI,EQ,6)  
/*  
// WLM99 EXEC PGM=IKJEFT01,REGION=0M,DYNAMNBR=50,PARM='%WLMPP'  
// SYSEXEC DD DISP=SHR,DSN=your.rexx.library  
// SMF  DD DISP=(SHR,KEEP),DSN=your.copied.by.sort.to.VB.smf.dataset  
// OUT99 DD DSN=sysuid.output.dataset,  
//         SPACE=(CYL,(30,15)),UNIT=SYSDA,  
//         DISP=(NEW,PASS),  
//         DCB=(RECFM=FB,LRECL=140)  
// SYSPRINT DD SYSOUT=*  
// SYSTSPRT DD SYSOUT=*  
// SYSTSIN DD DUMMY  
/*  
// RPT996 EXEC PGM=ICETOOL,REGION=0M  
// TOOLMSG DD SYSOUT=X  
// DFSMSG DD SYSOUT=X  
// REPORT DD SYSOUT=X  
// OUT99 DD DISP=(SHR,KEEP),DSN=sysuid.output.dataset  
// TEMP DD DSN=&&TEMPV,SPACE=(CYL,(15,15)),UNIT=SYSDA  
// TOOLIN DD *  
COPY FROM(OUT99) TO(TEMP) USING(SMFI)  
DISPLAY FROM(TEMP) LIST(REPORT)    -  
TITLE('WLM POSTPROCESSOR REPORT') DATE TIME -  
HEADER('TIME') ON(13,8,CH) -  
HEADER('SID') ON(23,4,CH) -  
HEADER('S.CLASS') ON(46,8,CH) -  
HEADER('PERIOD.') ON(55,1,CH) -  
HEADER('LOCAL PI') ON(57,3,CH) -  
HEADER('SYSPLEX PI') ON(62,3,CH) -  
HEADER('GOAL TYPE') ON(67,15,CH) -  
HEADER('GOAL VALUE') ON(83,3,CH) -  
HEADER('IMP') ON(87,1,CH) -  
HEADER('PERC.') ON(90,2,CH) -  
HEADER('CPU DP') ON(93,3,CH) -  
HEADER('IO DP') ON(97,3,CH) -  
BREAK(1,11,CH) -  
BTITLE('DAILY REPORT') -  
BLANK  
*/
//SMFICNTL DD *
* Example: select only peak period (ie 10 am - 2 pm)
* In a similar fashion one may construct customized INCLUDE statement
* and select other fields (ie LPI > 1, LPI > SPI...)
OPTION COPY
INCLUDE COND=(13,5,CH,GT,C'10:00',AND,13,5,CH,LT,C'13:59')
/*

WLMPP EXEC
/* REXX EXEC to read and format SMF records */
ADDRESS TSO

'EXECIO * DISKR SMF ( STEM x. FINIS'
do i = 1 to x.Ø

smftype = c2d(SUBSTR(x.i,2,1))   /* SMF record type */
smfstype = c2d(SUBSTR(x.i,19,2)) /* Record subtype */
/*-----------------------------------------------*/
/* Check SMF record type & subtype (ie 99.6)       */
/*-----------------------------------------------*/
IF smftype = '99' & smfstype = '6' THEN
  DO
    offset = c2d(SUBSTR(x.i,69,4))  /* Offset to period section */
    len = c2d(SUBSTR(x.i,73,2))    /* Length of period section */
    cpon = c2d(SUBSTR(x.i,75,2))   /* Number of period sections */
    /*-----------------------------------------------*/
    /* Unpack SMF date & decode SMF time             */
    /*-----------------------------------------------*/
    smfdate = SUBSTR(c2x(SUBSTR(x.i,7,4)),3,5) /* unpack SMF date */
    time = c2d(SUBSTR(x.i,3,4))     /* decode SMF time */
    time1 = time % 1000
    hh = time1 % 3600
    hh = RIGHT("0"||hh,2)
    mm = (time1 % 60) - (hh * 60)
    mm = RIGHT("0"||mm,2)
    ss = time1 - (hh * 3600) - (mm * 60)
    ss = RIGHT("0"||ss,2)
    smftime = hh||":"||mm||":"||ss           /* Compose SMF time*/
    /*-----------------------------------------------*/
    /* Process all class periods                    */
    /*-----------------------------------------------*/
    do j = 0 to cpon
      incr = (offset + (j*len)) - 3           /* Incremental position */
      sclass = SUBSTR(x.i,incr,8)            /* Class name */
      period = c2d(SUBSTR(x.i,incr+8,2))    /* Class period number */
      if period > '0' then do
        sysid = SUBSTR(x.i,11,4)          /* System identification */
      end
    end
  end
end
syslvl = SUBSTR(x.i,53,8) /* System level */
syname = SUBSTR(x.i,61,8) /* System name */
gt = c2d(SUBSTR(x.i,incr+10,1)) /* Goal type */
pct = c2d(SUBSTR(x.i,incr+11,1)) /* Goal percentile */
/
/* Reformat goal type values into goal description */
SELECT
  when gt=0 then goal='System/STC/Srv'
  when gt=1 then goal='Shr.Resp (sec.)'
  when gt=2 then goal='Lng.Resp (sec.)'
  when gt=3 then goal='Velocity (%)'
  when gt=4 then goal='Discretionary'
END

gval = c2d(SUBSTR(x.i,incr+20,4)) /* Goal value */
imp = c2d(SUBSTR(x.i,incr+24,2)) /* Period importance */
dp = c2d(SUBSTR(x.i,incr+26,1)) /* Dispatching prty. */
iodp = c2d(SUBSTR(x.i,incr+27,1)) /* I/O priority */
mpli = c2d(SUBSTR(x.i,incr+28,2)) /* MPL in-target */
mplo = c2d(SUBSTR(x.i,incr+30,2)) /* MPL out-target */
rua = c2d(SUBSTR(x.i,incr+32,4)) /* Number of ready ASIDs*/
pspt = c2d(SUBSTR(x.i,incr+36,4)) /* Time swapped out */
psitar= c2d(SUBSTR(x.i,incr+40,4)) /* Storage isolation */
lpi = c2d(SUBSTR(x.i,incr+44,4)) / 100 /* Local PI */
spi = c2d(SUBSTR(x.i,incr+48,4)) / 100 /* Sysplex PI */
sdata = c2d(SUBSTR(x.i,incr+52,4)) /* Offset to server sec.*/
slen = c2d(SUBSTR(x.i,incr+56,2)) /* Length of server sec.*/
snum = c2d(SUBSTR(x.i,incr+58,2)) /* Number of server ent.*/
/
/* Reformat goal value according to the goal type */
SELECT
  when gt=0 then gvvv='n/a'
  when gt=1 then gvvv=gval/1000
  when gt=2 then gvvv=gval/1000
  otherwise gvvv=gval
END

rec99 = left(Date('N',smfdate,'J'),11) left(smftime,9),
  left(sysid,4) left(syslvl,8) left(syname,8),
  left(sclass,8) left(period,1) left(lpi,4),
  left(spi,4) right(goal,15) right(gvvv,3),
  left(imp,2) right(pct,2) left(dp,3),
  left(iodp,3) right(mpli,2) right(mplo,2),
  right(rua,4) right(pspt,4) right(psitar,4),
  right(sdata,4) right(slen,2) right(snum,2)

PUSH rec99
"EXECIO 1 DISKW OUT99"
It is strongly recommended that this report be used in conjunction with the RMF postprocessor service class reports, which allow us to look further into those workloads that are not performing as expected. These reports provide more detailed information about specific service classes. The RMF service class period report is created using the SYSRPTS(WLMGL(SCPER)) control card with the RMF postprocessor. Another way to gain a quick glance at service class performance and various types of resource delays is by using the RMF postprocessor overview record control statements. In the example below, the service class TSO period 1 is being examined. One would need to add control cards to specify other periods (ie second and third period if applicable and identical control cards for any other service classes one wants to report on).

Example:

```plaintext
//RMFSTEP1 EXEC PGM=ERBRMFPP,REGION=ØM
//MFPINPUT DD DISP=SHR,DSN=SMF.SORTED,DATASET
//MFPMSGDS DD SYSOUT=*
FLICTS*******************************************************************************/
/*GOAL MODE INDICATORS for TSO class - PI, USING AND DELAY SAMPLES */
FLICTS*******************************************************************************/
//SYSIN DD *
SYSOUT(O)
NOSUMMARY
OVERVIEW(REPORT)
DATE(MMDDYYYY,MMDDYYYY)
ETOD(Ø800,1500)
OVW(PI(PI(S.TSO.1)),NOSYSTEMS)    /* PERFORMANCE INDEX, SC.PERIOD 1 */
OVW(ENDEDTRX(TRANSTOT(S.TSO.1)),NOSYSTEMS)/* ENDED TRANSACTIONS */
OVW(VELOCITY(EXVEL(S.TSO.1)),NOSYSTEMS) /* ACTUAL VELOCITY */
OVW(CPUUSING(CPUUSGP(S.TSO.1)),NOSYSTEMS) /* CPU USING % */
OVW(CPUDLYP(CPUDLYP(S.TSO.1)),NOSYSTEMS) /* CPU DELAY % */
OVW(IOUSING(IOUSGP(S.TSO.1)),NOSYSTEMS) /* I/O USING % */
OVW(IODELAY(IODELAYP(S.TSO.1)),NOSYSTEMS) /* I/O DELAY % */
OVW(APPLPERC(APPLPER(S.TSO.1)),NOSYSTEMS) /* APPLICATION % */
OVW(UNKNOWN(UNKP(S.TSO.1)),NOSYSTEMS)    /* UNKNOWN STATE % */
OVW(IDLE(IDLEP(S.TSO.1)),NOSYSTEMS)      /* IDLE STATE % */
```
Locating strings in files

For most purposes, the Searchfor utility is an acceptable way to search for strings within files. However, it lacks support for VSAM files and a way to limit the scope of the search. For example, I need to search for a name that should exist in a name field, but eventually might also exist in other fields, like address, etc. If I know the position of the field I want to search within each record, then I may wish to restrict my search to that area. Or I may choose to search only a limited set of records, and not the entire file.

To satisfy these needs, I wrote a search program that accepts both sequential and VSAM files, allows the search zone to be limited to a range of columns within each record or to a range of records, permits the search string to be specified in hexadecimal, and also permits case to be ignored when performing comparisons (this applies only to strings specified as characters and to standard a-z characters, but you can change the translation table within the program, if you like).

The application consists of an Assembler program and a front-end formed by a REXX EXEC and an ISPF panel. It looks like this:

```
+---------------- Locate a string in a file ---------------------+
|                                                                |
|    File......: AARCF3.TEST.BA3ØØ                             |
|                                                                |
|    Search for: x'Ø3f5f4c1a3                                   |
|       (Enter text string or begin with X' for hexadecimal)    |
|                                                                |
|       Ignore case - only valid for text (Y,N).: Y             |
```
In this example, we search for a hexadecimal string (since it begins with x', the ending quote is optional and can be omitted) with column boundaries 12 through to 25 and record boundaries 10,000 through to 15,000.

Since the string is hexadecimal, the ignore case field has no meaning. The program launches a job, and the result of the search can be found in sysprint: it states the record numbers where a match was found, the total number of record matches, and the total number of records searched:

String found in record number abrasive
String found in record number abrasive
String found in record number abrasive
Number of records where string found: 3
Number of records searched: 5

LOCATE ASSEMBLER PROGRAM

This program reads an input file and searches for a string in each record. The search can be limited to a column range within each record (Col1 to Col2) or to a set of records (Rec1 to Rec2). If flag = 'X', string is represented in hexadecimal. If flag = 'Y', string and searched areas are uppercased before
* comparison takes place (case is ignored). This applies only to  *
* a-z standard EBCDIC characters.                          *
*====================================================================*
&PROGRAM SETC  'LOCATE'
&PROGRAM AMODE 31
&PROGRAM RMODE 24
&PROGRAM CSECT
SAVE (14,12)
LR   R12,R15
USING &PROGRAM,R12
ST   R13,SAVEA+4
LA   R11,SAVEA
ST   R11,8(R13)
LR   R13,R11
B    GETPARMS
DC   CL16' &PROGRAM 1.1'
DC   CL8'&SYSDATE'

*====================================================================*
* Check and validate parameters                                     *
*====================================================================*
* GETPARMS DS ØH
LR   R2,R1                Copy parm pointer to R2.
L    R2,Ø(Ø,R2)          Load parm address
LH   R3,Ø(R2)            Load parm length in R3
OPEN  (SYSPRINT,OUTPUT)   Open sysprint (for error msgs)
LTR  R3,R3                Any parm entered?
BZ    EXIT1               No, error
  LR   R6,R2
  AR   R6,R3              R6: point after end of parms
  LA   R6,2(Ø,R6)        Skip 2 bytes of parmlength
  LA   R2,2(Ø,R2)        Clear length counter
* CONVERT  EQU  *
LA   R9,3                4 byte length parms
EX   R9,EXPACK           Execute pack
CVB  R7,PARMPACK         Convert to binary into R7
S    R7,=F'1'            Turn limit to offset
ST   R7,PARM1            And store it
* LA   R2,4(Ø,R2)        Inc parm pointer
EX   R9,EXPACK           Execute pack
CVB  R7,PARMPACK         Convert to binary into R7
ST   R7,PARM2            And store it
* LA   R2,4(Ø,R2)        Inc parm pointer
LA   R9,7                8 byte length parms
EX   R9,EXPACK           Execute pack
CVB R7,PARMPACK  Convert to binary into R7
ST R7,PARM3  And store it

*  
LA R2,8(Ø,R2)  Inc parm pointer
EX R9,EXPACK  Execute pack
CVB R7,PARMPACK  Convert to binary into R7
ST R7,PARM4  And store it

*  
LA R2,8(Ø,R2)  Inc parm pointer
MVC FLAG,Ø(R2)  Store flag parameter

*  
LA R2,1(Ø,R2)  Inc parm pointer to string
SR R6,R2  Length of string to search
SH R6,=H'1'  Length of string to search
EX R6,EXMOVE  Move to string
STH R6,STRINGL  Keep string length (-1)
CLI FLAG,C'X'  Hexadecimal string?
BNE VALIDPR  No, jump ahead

*  
MVC STR1,STRING1  Convert string to real hexadecimal
MVC STR2,STRING2  characters in three 12 byte parts
MVC STR3,STRING3
NC STR1,XAND
TR STR1,XTRN
NC STR2,XAND
TR STR2,XTRN
NC STR3,XAND
TR STR3,XTRN
PACK STRP1,STR1(13)
PACK STRP2,STR2(13)
PACK STRP3,STR3(13)
MVC STRING(6),STRP1
MVC STRING+6(6),STRP2
MVC STRING+12(6),STRP3

*  
LH R6,STRINGL  Get original length (-1)
LA R6,1(Ø,R6)  Add 1
SRL R6,1  Divide length by two
SH R6,=H'1'  Ready for comparisons (-1)
STH R6,STRINGL  Store it

*  
VALIDPR EQU *  Validate parameters
CLC PARM1,PARM2
BH ERRMSG1
CLC PARM3,PARM4
BH ERRMSG2
L R7,PARM2
S R7,PARM1
CR R6,R7
BH ERRMSG3

CLI FLAG,C'Y'         Ignore case specified?
BNE OPENACB1          No, jump ahead
LH R1Ø,STRINGL        Load string length (-1)
LA R1Ø,1(Ø,R1Ø)       Reset correct length
LA R5,STRING          Execute uppercase translation

*====================================================================*
* Check whether file is VSAM or sequential. If VSAM, check for ESDS *
*====================================================================*

OPENACB1 EQU *          Open ACB for VSAM input file
OPEN INFILEA            If error, go open the file as
LTR R15,R15             sequential.
BNZ OPENDCB1
TESTCB ACB=INFILEA,     File is VSAM, check for ESDS
 ATRB=ESDS
BNE READFILE

ESDSFIL1 EQU *          File RPL for ESDS
MODCB RPL=INFILER,
 OPTCD=ADR
B READFILE

OPENDCB1 EQU *          Open sequential file
OPEN (INFILED,INPUT)
LTR R15,R15
BNZ ERRMSG4
MVI FILETYP1,C'S'       Set flag sequential
LA R2,INFILED           R2: IHADCB of input file
USING IHADCB,R2

*====================================================================*
* Read loops (VSAM loop or sequential loop) and compare subroutine *
*====================================================================*

READFILE EQU *          *
XR R7,R7                 Record counter
L R8,PARM3               First record
L R9,PARM4               Last record
CLI FILETYP1,C'V'        VSAM file?
BNE LOOPSEQ              No, go to sequential

LOOPVSA EQU *            VSAM loop
LA R7,1(Ø,R7)           Increment record counter
GET RPL=INFILER         Read VSAM file
LTR R15,R15             End of file?
BNZ EXITØ
CR R7,R8                First record attained?
BL    LOOPVSA          No, read again
CR    R7,R9               Last record attained?
BH    EXITØ             Yes, exit

*  
L    R4,VAREA1           Get address of data in R4.
SHOWCB  RPL=INFILER,
        AREA=LRECL1,
        LENGTH=4,
        FIELDS=RECLEN
L    R3,LRECL1           Get record length in R3
BAL   R1Ø,COMPARE      Call compare
B    LOOPVSA

*  
LOOPSEQ  EQU   *                   Sequential loop
LA    R7,1(Ø,R7)          Increment record counter
GET   INFILED           Read sequential
LR    R4,R1               R4: address of record
CR    R7,R8               First record attained?
BL    LOOPSEQ            No, read again
CR    R7,R9               Last record attained?
BH    EXITØ             Yes, exit
LH    R3,DCBLRECL        Load R3 with record length.
BAL   R1Ø,COMPARE      Call compare
B    LOOPSEQ

*  
COMPARE  EQU   *                   Compare subroutine
L    R5,PARM1            Left limit (offset)
L    R6,PARM2            Right limit
CR    R3,R6               Record smaller than last position?
BNL   COMPAREØ            No, continue.
LR    R6,R3               Yes, switch limit to record limit

*  
COMPAREØ  EQU   *               R6: length of searchable area
SR    R6,R5               R6: position to compare
AR    R5,R4               R6: length of searchable area
CLI   FLAG,C'Y'           Ignore case specified?
BNE   COMPARE9            No, jump ahead
EX    R6,EXTRAN          Execute uppercase translation

*  
COMPARE9  EQU   *               R6: number of searches in the line
SH    R6,STRINGL         R6: number of searches in the line
C    R6,=F'Ø'            If R6<Ø, string is greater than
BNH   COMPARE3            search area, so skip this record

*  
COMPARE1  EQU   *               length of search string
LH    R11,STRINGL        length of search string
EX    R11,EXCOMPAR      Execute compare
BNE   COMPARE2            If strings not equal, exit
LR    RØ,R7
BAL   R11,UNPACK
MVC MSGFND2,OUT1Ø
LR R0,R9
PUT SYSPRINT,MSGFND String found, send message
L R11,TOTFOUND Inc rec found counter
LA R11,1(Ø,R11)
ST R11,TOTFOUND
B COMPARE3 And return

COMPARE2 EQU *
LA R5,1(Ø,R5) Increment compare position
BCT R6,COMPARE1 Loop to next

COMPARE3 EQU *
BR R1Ø return

*====================================================================*
* Send final messages, close files, and exit                         *
*====================================================================*

EXITØ EQU *
L R0,TOTFOUND
BAL R11,UNPACK
MVC MSGTOT2,OUT1Ø
PUT SYSPRINT,MSGTOT
SR R7,R8
LR R0,R7
BAL R11,UNPACK
MVC MSGFIM2,OUT1Ø
PUT SYSPRINT,MSGFIM

EXIT1 EQU *
CLOSE INFILED
CLOSE INFILEA
CLOSE SYSPRINT
L R13,SAVEA+4
LM R14,R12,12(R13)
XR R15,R15
BR R14

*====================================================================*
* Other subroutines, execute instructions and work areas             *
*====================================================================*

EXCOMPAR EQU *
CLC Ø(Ø,R5),STRING

EXMOVE EQU *
MVC STRING,Ø(R2)

EXPACK EQU *
PACK PARMPACK,Ø(Ø,R2)

* EXTRAN EQU *
  TR Ø(Ø,R5),TRTAB

* UNPACK EQU *
  CVD RØ,REGDECIM
  UNPK OUT12,REGDECIM
  BR R11

* ERRMSG1 EQU *
  PUT SYSPRINT,MSG1
  B EXITØ

* ERRMSG2 EQU *
  PUT SYSPRINT,MSG2
  B EXITØ

* ERRMSG3 EQU *
  PUT SYSPRINT,MSG3
  B EXITØ

* ERRMSG4 EQU *
  PUT SYSPRINT,MSG4
  B EXITØ

* MSG1 DC CL8Ø'Error: Parm2 smaller than parm1'
* MSG2 DC CL8Ø'Error: Parm3 smaller than parm4'
* MSG3 DC CL8Ø'Error: String length smaller than search interval'
* MSG4 DC CL8Ø'Error opening input file'
* MSG5 DC CL8Ø'Record smaller than compare position'
* MSGFND DS ØCL8Ø
  MSGFND1 DC C'String found in record number '
  MSGFND2 DC CL5Ø' '
  MSGTOT DS ØCL8Ø
  MSGTOT1 DC C'Number of records where string found: '
  MSGTOT2 DC CL5Ø' '
  MSGFIM DS ØCL8Ø
  MSGFIM1 DC C'Number of records searched. . . . . : '
  MSGFIM2 DC CL5Ø' '

* STRINGL DS H
  DS CL2
  STRING DS ØCL36
  STRING1 DS CL12
  STRING2 DS CL12
  STRING3 DS CL12
  DS CL4
  STR1 DS CL12
  DC C' '
LOCATE REXX EXEC

/* REXX MVS */
/* LOCATE - Locates a string within a file. */
/* Optional argument: file to search. */
/* */
/* This application consists of this EXEC, LOCATE ISPF panel, */
/* and LOCATE Assembler program. The load module should reside */
/* in the library indicated by the loablib variable below. */
/* */
/*---------------------------------------------------------------*/
arg file.
file = strip(file,\"\")
loadlib = "loadlib.with.locate.module"
tempfile = userid()||\".TEMP.FILE\"
I = "Y"
do forever
  address ispexec
  'addpop row(1) column(1)'
  'display panel(locate)'
  if rc = 8 then exit
  'rempop'
  address tso
  msg = ""
  hexastring = Ø
  if col1 = "" then col1 = 1
  if col2 = "" then col2 = 9999
  if rec1 = "" then rec1 = 1
  if rec2 = "" then rec2 = 99999999
  if col2 < col1 then do
    msg="Column2 cannot be smaller than column1"
iterate
end
if rec2 < rec1 then do
    msg="Record2 cannot be smaller than record1"
    iterate
end
str = strip(str)
lstr= length(str)
if left(str,2) = "x'" | left(str,2) = "X'" then do
    hexastring = 1
    str = strip(str,"T","'")
    str = translate(substr(str,3))
lstr= length(str)
    if datatype(str,"X") <> 1 then do
        msg="Invalid hexadecimal string"
        iterate
    end
lok = lstr // 2
if lok <> Ø then do
    msg="Odd number of characters in hexadecimal string"
    iterate
end
lstr = lstr / 2
end
if lstr > col2 - col1 + 1 then do
    msg="String is longer than column search zone"
    iterate
end
if msg = "" then leave
end
if hexastring = 1 then I = 'X'
parm = "'"right(col1,4,"Ø") || right(col2,4,"Ø") ||,
      right(rec1,8,"Ø") || right(rec2,8,"Ø") ||,
      I || str"'"
xx = msg(off)
"free dd (temp1)"
"alloc da('"tempfile"') dd(temp1) new reuse blksize(8000),
  lrec(80) recfm(f,b) dsorg(ps) space(1 1) tracks delete"
if rc <> Ø then do
    say "Error "rc" allocating" tempfile
    exit
end
queue "//userid()"Ø JOB LOCATE,MSGCLASS=X,CLASS=A"
queue ""//STEPØ EXEC PGM=LOCATE,"
queue ""// STEPLIB DD DISP=SHR,DSN="loadlib"
queue ""// INFILE DD DISP=SHR,DSN="file"
LOCATE ISPF PANEL

)ATTR
  _ TYPE(INPUT) CAPS(ON) JUST(LEFT) COLOR(RED)
  $ TYPE(INPUT) CAPS(OFF) JUST(LEFT) COLOR(RED)
  # TYPE(INPUT) CAPS(ON) JUST(RIGHT) COLOR(RED)
  ? TYPE(TEXT) INTENS(HIGH) SKIP(ON) COLOR(PINK)
  % TYPE(TEXT) INTENS(HIGH) SKIP(ON) COLOR(YELLOW)
  + TYPE(TEXT) INTENS(LOW) SKIP(ON) COLOR(GREEN)
  ! TYPE(OUTPUT) CAPS(OFF) SKIP(ON) COLOR(WHITE)
)BODY WINDOW(70,17)
  + ? File.......:_FILE          +
  + ? Search for:$STRI          +
  +     (Enter text string or begin with X' for hexadecimal)
  +     Ignore case - only valid for text (Y,N).:_I
  + % Search columns             Search records
  +     First column.#COL1+      First record.#REC1 +
  +     last column.#COL2+      Last record.#REC2 +
  + !MSG Enter - execute         PF3/15 cancel
)INIT
&ZWINTTL = 'Locate a string in a file'
)PROC
&ver='Y,N'
VER(&FILE,nonblank,dsname)
VER(&STRI,nonblank)
VER(&STRI,nonblank)
VER(&I,NONBLANK,listv,&ver)
VER(&COL1,num)
VER(&COL2,num)
VER(&REC1,num)
VER(&REC2,num)
)END

Systems Programmer (Portugal) © Xephon 2005
Boosting VSAM performance with SMB

Ever since its introduction some 30 years ago, VSAM has been a popular and reliable data storage construct on MVS systems. VSAM is still the cornerstone of on-line applications such as IMS and CICS, and is widely used in ISV packages and in-house-written batch applications. However, with 24x7 operation becoming a necessity, batch windows must shrink in order to lessen their impact on on-line systems. The most effective way to cut down the batch window is to optimize I/O, and this article examines the results of a sample tuning exercise. Of course, it is a well-known fact that the fastest I/O is the one that is never issued. That is to say, an application that is having I/O performance problems will perform better if we can cut down on the number of I/Os. The whole concept of SMS automatic block sizes according to DASD device was to reduce the number of I/Os for that specific DASD device type. Similarly, for VSAM data, automatic effective buffering can significantly reduce the number of I/Os, response time, and elapsed job time, thereby improving application performance.

It is well known that VSAM is very important to most installations, yet it is rarely utilized optimally. One consequence of this is that jobs accessing VSAM files almost always run longer than necessary. Thus, tuning native VSAM datasets is still an important part of the overall tuning process at many installations. Almost certainly, the largest performance gains can be achieved with good VSAM buffering – it is in fact the single most important aspect of VSAM tuning and will achieve the biggest performance boost. If implemented correctly, these buffering methodologies will greatly reduce disk I/Os, reduce CPU time, and lead to better job turnaround time. Now, with the advent of System Managed Buffering (SMB), high performance can be achieved through standard OS/390 system interfaces, with virtually no application programmer effort, and with no JCL changes. System Managed Buffering is a feature of DFSMSdfp, directed at support for batch application processing, and is intended as a means of achieving two things. The first one is to update the current defaults for processing VSAM datasets. This is necessary in order to utilize
current hardware technology to effect the processing of VSAM data. The second one is to initiate a buffering technique, other than that specified by the application program, that would improve application performance.

SYSTEM MANAGED BUFFERING

Before we see how SMB works and how you can take advantage of it, it might be useful to understand the overall buffering picture for VSAM files. There are two ways of addressing buffering of VSAM data offered by OS/390 and z/OS. The first method uses Non-Shared Resources (NSR), where buffers are dedicated to the processing of a single VSAM file. NSR means that each VSAM file in a task will have its own dedicated buffers assigned within the program address space, and, hence, will not share them with any other VSAM file that is open within the task. NSR is also the automatic default type of VSAM buffering logic. On the other hand, using Logically Shared Resources (LSR) allows the sharing of buffer pools among multiple VSAM files. While both NSR and LSR can be defined within an application, there is a significant difference when it comes to how an application maximizes its use of these buffer pools to process data – sequential processing of data works best using NSR and random data processing works best using LSR. Does this mean that applications need to be changed? Sometimes they need changing. Moreover, sometimes it is necessary to understand the type, access method, format, and options of the file. For example, do dataset options call for key access (KEY), sequential access (SEQ), addresses access (ADR), or access to CI (CNV)?

System Managed Buffering (SMB) for VSAM datasets is a fairly new facility introduced with DFSMS Version 1.4 for KSDS files only. This was enhanced with DFSMS 1.5 to include all types of VSAM file. Basically, the system decides how many buffers to use for data and index portions (the case of NSR) or buffer pools size (the case of LSR), with four basic buffer allocation algorithms that can be chosen or specified:

- Direct Optimized (DO) – SMB optimizes for totally random record access. This is appropriate for applications that access
records in a dataset in totally random order. This technique will override the user specification for using NSR buffering with an LSR buffering implementation. Random-access VSAM processing is automatically directed to use LSR, which will eliminate buffer stealing, exploit look-aside processing, ESA hiperspaces, and in-core indexes. The DO technique is elected if the ACB specifies only the MACRF=(DIR) option for accessing the dataset. If either SEQ or SKP are specified, in combination with DIR or independently, DO is not selected. The selection can be overridden by the user specification of ACCBIAS=DO on the AMP=parameter of the associated DD statement. Note should be taken of the fact that the MACRF type of access is just an intention. The real type of access is declared per I/O operation in the RPL.

- **Direct Weighted (DW)** – SMB optimizes for mixed-mode processing (both direct and sequential), but ‘weights’ the buffer allocations for key-direct. This will provide minimum read-ahead buffers for sequential retrieval and maximum index buffers for direct requests. The size of the dataset is a minor factor in the storage that is required for buffering. This technique requires approximately 100KB of processor storage for buffers, with a default of 16MB.

- **Sequential Optimized (SO)** – SMB optimizes for sequential processing. It is appropriate for applications reading the entire dataset from the first to last record or a large percentage in sequential order. The size of the dataset is not a factor in the processor virtual storage that is required for buffering. Approximately 500KB of processor virtual storage, defaulted to above the 16MB line, is required for buffers for this technique.

- **Sequential Weighted (SW)** – SMB optimizes for mixed-mode processing (both direct and sequential), but ‘weights’ the buffer allocations for sequential. It will use read-ahead buffers for sequential and provide additional index buffers for direct requests. The read-ahead will not be the large amount of data transferred as with SO. The size of the dataset is a minor factor in the amount of processor virtual storage that buffering requires.
This technique requires approximately 100KB of processor virtual storage for buffers, with the default above 6MB.

General discussion and guidelines related to processing with each technique are fully documented in *VSAM Demystified* (SG24-6105).

The change-over to SMB is easy enough – it can be simply done by defining an extended format dataset through an SMS data class with RECORD_ACCESS_BIAS=SYSTEM/USER. Or, if you prefer JCL changes, it can be invoked in a specific job stream by specifying ACCBIAS on the AMP parameter for the dataset’s DD statement.

In the first case, the technique that will be defaulted to by the system is based on the application specification for the type of access intention ACB MACRF=(DIR,SEQ,SKP) and influenced by the specifications in the associated Storage Class (SC) for direct millisecond response, direct bias, sequential millisecond response, and sequential bias.

In the second case, the technique is externally specified by using the ACCBIAS JCL subparameters of the AMP DD parameter – probably the easiest and best option is the ACCBIAS=SYSTEM option. You can specify ACCBIAS equal to one of the following values:

- **USER** – bypass SMB. This is the default if you code no specification for the ACCBIAS subparameter. This default is not used when the data class specifies RECORD_ACCESS_BIAS.
- **SYSTEM** – force the system to determine the buffering technique.

One can also explicitly request a specific buffer allocation algorithm by specify the SMB buffer processing as SO/SW/DO/DW. One of the problems with SMB arises in situations where you have a batch program that does skip-sequential, sequential, and random processing all in the same run. In many such cases, that we have seen it’s often been a good compromise just to default to ACCBIAS=SYSTEM. For a detailed description of each AMP option see *MVS: JCL Reference* (SA22-7597).
During a testing phase we turned on Systems Managed Buffering (through DATACLASS) for a large VSAM file, but in order to see how SMB works, as well as to prevent production problems, we decided to bypass SMB processing by specifying RECORD_ACCESS_BIAS=USER and later on we used JCL’s AMP parameter ACCBIAS=SO (see below):

VSAM file buffers & buffering management

<table>
<thead>
<tr>
<th>Records</th>
<th>Access:</th>
<th>Buffers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job</td>
<td>Run date</td>
<td>Elapsed time</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>---------------</td>
</tr>
<tr>
<td>MYJOB</td>
<td>17 Oct 2003 00:49:24:46</td>
<td>PROD.HISTFILE.DATA</td>
</tr>
<tr>
<td>(1)</td>
<td>MYJOB</td>
<td>17 Oct 2003 00:49:24:46</td>
</tr>
<tr>
<td>MYJOB</td>
<td>18 Oct 2003 00:42:45:09</td>
<td>PROD.HISTFILE.DATA</td>
</tr>
<tr>
<td>MYJOB</td>
<td>18 Oct 2003 00:42:45:09</td>
<td>PROD.HISTFILE.INDEX</td>
</tr>
<tr>
<td>MYJOB</td>
<td>19 Oct 2003 00:50:07:19</td>
<td>PROD.HISTFILE.DATA</td>
</tr>
<tr>
<td>(2)</td>
<td>MYJOB</td>
<td>19 Oct 2003 00:50:07:19</td>
</tr>
<tr>
<td>MYJOB</td>
<td>21 Oct 2003 00:50:57:52</td>
<td>PROD.HISTFILE.DATA</td>
</tr>
<tr>
<td>MYJOB</td>
<td>21 Oct 2003 00:50:57:52</td>
<td>PROD.HISTFILE.INDEX</td>
</tr>
<tr>
<td>MYJOB</td>
<td>22 Oct 2003 00:26:22:79</td>
<td>PROD.HISTFILE.DATA</td>
</tr>
<tr>
<td>(3)</td>
<td>MYJOB</td>
<td>22 Oct 2003 00:26:22:79</td>
</tr>
<tr>
<td>MYJOB</td>
<td>23 Oct 2003 00:26:41:31</td>
<td>PROD.HISTFILE.DATA</td>
</tr>
<tr>
<td>MYJOB</td>
<td>23 Oct 2003 00:26:41:31</td>
<td>PROD.HISTFILE.INDEX</td>
</tr>
</tbody>
</table>

Notes:

(1) Job access statistics before converting dataset to extended format.
(2) Dataset converted to extended format with Rec_Acc_Bias=USER (bypass SMB).

(3) JCL AMP parameter override of data class definition (ACCBIAS=SO).

The order of precedence for specifying values that decide if and how SMB will be invoked is this: JCL specifications, then the data class Record_Access_Bias parameter, then the storage class parameters, then the MACRF values. That is, whatever is specified in the JCL will always take precedence. This also means that one may wish to tell a lie to VSAM about intent (for example direct versus sequential processing) and SMB will be fooled. Because SMB is not taking any sample of behaviour, it relies on the access intent of the OPEN. However, telling a lie is not a wise thing to do: incorrect use of a buffering strategy will result in a significant increase in I/O, thus causing long-running batch jobs and poor performance (see below):

<table>
<thead>
<tr>
<th>Buffering</th>
<th>EXCPs</th>
<th>Clock time (min)</th>
<th>CPU time (sec)</th>
<th>CONN (k)</th>
<th>Buffers (D/I)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSR - Default</td>
<td>402472</td>
<td>41.4</td>
<td>251.16</td>
<td>1099</td>
<td>4 2</td>
</tr>
<tr>
<td>ACCBIAS=SO</td>
<td>34991</td>
<td>26.0</td>
<td>233.66</td>
<td>918</td>
<td>49 4</td>
</tr>
<tr>
<td>ACCBIAS=DO</td>
<td>793868</td>
<td>45.0</td>
<td>294.28</td>
<td>1342</td>
<td>0 0</td>
</tr>
</tbody>
</table>

Gain using SMB (%): 91.3 37.19 9.96 13.73

(DO vs. Default): 97.2 8.7 17.16 22.11

SMB RESTRICTIONS AND POTENTIAL PROBLEMS

There are two main restrictions to SMB. The first one is that SMB support is currently limited to extended format VSAM files that use NSR buffering. To be in extended format, the dataset must be system managed (SMS) and use a data class defined with DSNTYPE=EXT. On the other hand, SMB will get involved only when NSR buffering is specified by the application program, ACB MACRF=(NSR). It will not get involved with the MACRF parameters RST (ACB reset option), UBF (USER buffering), GSR (Global Shared Resources), LSR (Local Shared Resources), RLS (Record Level Sharing), or ICI (Improved Control Interval processing). For releases prior to z/OS 1.3 DFSMS, processing the dataset through the alternate index of
the path specified in the DDname is not supported. When the conditions above are not satisfied, the job does not abend, but the SMB services are not used and no messages are issued.

The second restriction is that SMB is invoked at dataset open processing only: after the initial decision is made during that process, SMB has no further involvement.

Thus far two basic storage-related problems have emerged, especially regarding the use of the ACCBIAS=DO option. SMB ACCBIAS=DO is in fact equivalent to BLSR in that, in both cases, VSAM LSR buffer pools are built for each dataset opened with this technique in a single application program. The size of the pool is based on the actual dataset size at the time the pool is created. A separate pool is built for both data and index components, if applicable, for each dataset. There is no capability for a single pool to be shared by multiple datasets. The index pool is sized to accommodate all records in the index component. The data pool is sized to accommodate approximately 20% of the user records in the dataset. This also means that the processor virtual storage requirement will increase with each OPEN after records have been added and the dataset has been extended beyond its previous size. Thus, for very large VSAM KSDS files, a program or job step might abend with ACCBIAS=DO because of storage problems unless SMB’s default options regarding buffer pool allocations are overridden.

Again, two options are available to tackle this problem. Increasing the job’s region size to support the buffers (think multiple megabytes just for the buffers) might avoid abends. Then again, it might not help, as was the case with a very large VSAM KSDS file we were testing, even though we had increased the job’s region size to the maximum possible.

On the other hand, the use of the SMBVSP parameter on the AMP=parameter (not present in the data class specification) can alleviate the storage impact since it restricts the amount of virtual storage to be obtained for buffers when opening the dataset. It is used to override the default buffer space to be obtained, which is calculated assuming that 20% of the data accounts for 80% of the accesses. The buffer space acquired is split across two LSR pools –
one for the index and one for the data.

There is also an additional AMP parameter that can be used in conjunction with the SMBVSP parameter, and it can help to reduce the storage problems. The SMBHWT parameter can be used to provide buffering in hiperspace in combination with virtual buffers for the data component. These buffers may be allocated for the base data component of the sphere. If the CI size of the data component is not a multiple of 4KB, both virtual space and hiperspace are wasted. It can be specified as an integer from 0 to 99. The value specified acts as a weighting factor for the number of hiperspace buffers to be established. This can reduce the size required for an application region, but does have implications related to processor cycle requirements. That is, all application requests must orient to a virtual buffer address. If the required data is in a hiperspace buffer, the data must be moved to a virtual buffer after ‘stealing’ a virtual buffer and moving that buffer to a Least Recently Used (LRU) hiperspace buffer.

Finally, if the optimum amount of storage required for this option is not available, SMB will reduce the number of buffers and retry the request. The retry capability for the DO technique was added in z/OS 1.3 DFSMS. For data, SMB will make two attempts, with a reduced amount and a minimum amount. For an index, SMB reduces the amount of storage only once, to a minimum amount. If all attempts fail, the DW technique is used. The system issues an IEC1611 message to advise that this has happened.

If you are running a 24-bit program (amode=rmode=24) be aware that the storage for buffers for SMB techniques are obtained above 16 megabytes (above the line), and in order to prevent problems IBM recommends that RMODE31=NONE be specified on the AMP= parameter for those datasets using SMB.

IDENTIFYING JOBS THAT MIGHT BENEFIT FROM SMB

The jobs that might benefit from SMB are those with certain application characteristics, most important of which are a data reference pattern and options specified by the application program
(ACB MACRF). The best candidates are long-running jobs as well as jobs with a high execute channel program (EXCP) count.

SMF type 64 records are probably used more frequently than any other data source for tuning VSAM applications. Using these records you can identify the programs with the highest amount of VSAM activity (such as number of EXCPs, retrievals, inserts, deletes, CI and CA splits, insert strategy), analyse the effectiveness of buffer usage, and determine whether the dataset is being used concurrently by other jobs or tasks. To determine candidates for SMB, we have used SMF type 64 records to obtain information about the SMB candidate’s processing characteristics, including jobname, cluster/component name, change in number of EXCPs, and ACB MACRF fields. In addition, SMF type 64 records indicate whether a reduced or minimum amount of resource is being used for a data pool and whether DW is used. Bits 5–7 of SMF64RSC, which were previously reserved, are used to give more information about Direct Optimization (DO).

A detailed description of the layout of the SMF type 64 record can be obtained from the MVS System Management Facilities (SMF) (SA22-7630) manual. One can also find the type 64 subtype descriptions in macro IDASMF64 in SYS1.MACLIB.

CODE
Based on record descriptions obtained from the above mentioned manual, a sample SMB report writer was written. The code is a two-part stream. In the first part (COPYSMF) selected SMF records (selection being defined by INCLUDE’s condition) are copied from the SMF dataset to a file that can be used as a base of archived records. In the second part, SMB64, the captured records are formatted by invoking SMB EXEC and two reports are produced.

Each report consists of two sets of variables. The first set is a fixed one consisting of the variables that uniquely identify the VSAM file or job being monitored. This set is meant to be used across all reports. The pool of variables in this set contains generated observation number, job name, date stamp, dataset allocation elapsed
time, cluster/component name, total number of records, number of records retrieved in a job run, and number of EXCPs. The second set of printed variables is area specific and pertains only to the VSAM file performance domain being monitored. Note should be taken of the fact that elapsed time in these reports is not the execution clock time (wall time) that we are accustomed to thinking of. This ‘elapsed’ time in fact represents the length of time the file was kept open (for details see APAR OW43854).

The first report shows standard VSAM file attributes and processing activity as well as the type of access to the record – key, rba (relative byte addresses) or cnv(access the dataset by control interval), dataset addressability, and format. As already stated, there are some restrictions when considering the use of SMB. This report shows whether there are any restrictions – user buffering, ICI processing, alternate index, NSR required, and/or if extended format is required.

The second report is a VSAM file buffer management report and it provides buffering-related information such as number of buffers used per component (system determined or user defined), buffer space, addressing mode for buffers (24/31 mode), as well as whether or not the buffers have been fixed in real storage. The more interesting part of the report provides answers to questions like: Is there any method to find out whether SMB gets invoked at all? Wouldn’t it be nice not only to know that SMB is invoked but also how much (and what) it does to the job (or datasets). This report provides the answer to these two questions by means of SMB-related information. Was SMB invoked at all? (no, yes: by JCL or SYSTEM); which optimization technique was used? (DO, DW, SO, SW, or none); and in conjunction with that, what data reference pattern was used: sequential access (records were requested in either ascending or descending sequence), direct access (records were randomly requested), skip sequential (records were processed in sequence but some records may have been skipped), or a combination of these? In the case of the Direct Optimized (DO) technique, additional indicators are available, such as the amount of virtual storage set by the SMBVSP parameter, whether hiperspace buffers were used, whether insufficient virtual storage problems occurred, indicators of whether a reduced or minimum amount of resource is
being used for a data pool, and whether DW is used (the case of retry technique).

**SMBJOB**

```plaintext
//DEL       EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=X
//SYSIN     DD *
  DELETE hlq.SMF64.DATA
  SET MAXCC=Ø
/
//COPYSMF  EXEC PGM=ICETOOL
//TOOLMSG  DD SYSOUT=*  
//DFSMSG   DD SYSOUT=*  
//RAWSMF  DD DSN=your.smf.dataset,DISP=SHR
//SMF64    DD DSN=hlq.SMF64.DATA,
  // SPACE=(CYL,(x,y)),UNIT=SYSDA,
  // DISP=(NEW,CATLG,KEEP),
  // DCB=(RECFM=VB,LRECL=32756,BLKSIZE=3276Ø)
//TOOLIN   DD *
COPY FROM(RAWSMF) TO(SMF64) USING(SMFI)
//SMFICNTL DD *
  OPTION SPANINC=RC4,VLSHRT
  INCLUDE COND=(6,1,BI,EQ,64,AND,43,1,BI,NE,X'2Ø') * copy SMF 64
/
//SMB64    EXEC PGM=IKJEFTØ1,REGION=ØM
//SYSEXEC  DD DISP=SHR,DSN=your.rexx.library
//SMF64    DD DISP=SHR,DSN=hlq.SMF64.DATA
//SYSPRINT DD SYSOUT=*  
//SYSTSIN  DD *
prof nopref
%SMB
/*
```

**SMB EXEC**

```plaintext
/* REXX EXEC to read SMF 64 records - VSAM Component/Cluster Status */
signal ON ERROR

/*-----------------------------------------------*/
/* Part 1: Handle file allocation & dataset existence and */
/*         print report header and labels */
/*-----------------------------------------------*/
Address TSO
  userid=SYSVAR(SYSUID)
  r64fa =userid||'.r64fa.rep' /* File processing/attribute*/
  r64bf =userid||'.r64bf.rep' /* Buffering report */
  x = MSG('OFF')
```
IF SYSDSN(r64fa) = 'OK'
THEN "DELETE "r64fa" PURGE"
IF SYSDSN(r64bf) = 'OK'
THEN "DELETE "r64bf" PURGE"
"ALLOC FILE(S64FA) DA("R64FA")",
"UNIT(SYSALLDA) NEW TRACKS SPACE(90,30) CATALOG",
"REUSE RELEASE LRECL(245) RECFM(F B)"
"ALLOC FILE(S64BF) DA("R64BF")",
"UNIT(SYSALLDA) NEW TRACKS SPACE(90,30) CATALOG",
"REUSE RELEASE LRECL(205) RECFM(F B)"
fi.1 =left( ' ',8, ' '),
||'VSAM file processing & attribute report'||left(' ',15, ' ')
fi.2 = '
fi.3 =left( ' ',8, ' ')||'Report produced on',
||left( ' ',1, ' ')||left(date(),11),
||left( ' ',1, ' ')||left('at ',3, ' ')||left(time(),10)
fi.4 = '
fi.5 = left( ' ',83, ' ')||left('# of ',4)||left(' ',2, ' '),
||left('Records in this run:',20)||left( ' ',10, ' '),
||left('Access by:',16)||left('CI',2),
||left( ' ',5, ' ')||left('Index',5),
||left( ' ',6, ' ')||left('Split -- Insert',20),
||left( ' ',8, ' ')||left('Data set:',9),
||left( ' ',9, ' ')||left('Res.',4)
fi.6 = left('obs',3) right('Job name',8) left('Run date',11),
left('Elapsed time',14) left('Cluster/Component name',30),
right('Excp',6) right('Records',9) right('ret'ved',8),
left('delete  insert update',21) right('Key',3),
right('Rba',3) right('Cnv',3) right('ICI',3),
left('Recl',4) right('kl',3) right('size',4),
right('level',8) right('CI',7) right('CA',9),
center('strategy',8) right('ext.',5),
left('Address',8) left('Format',8) left('sharing',7),
left('Restriction',12)
fi.7 = left('-',242,-)
"EXECIO * DISKW s64fa (STEM fi.)"
bf.1 =left( ' ',8, ' '),
||'VSAM file buffers & buffering management'||left(' ',15, ' ')
bf.2 = '
bf.3 =left( ' ',8, ' ')||'Report produced on',
||left( ' ',1, ' ')||left(date(),11),
||left( ' ',1, ' ')||left('at ',3, ' ')||left(time(),10)
bf.4 = left( ' ',76, ' ')||left('# of Records',13),
||left( ' ',8, ' ')||left('Access:',8)||left(' ',8, ' '),
||left('Buffers:',8)||left( ' ',30, ' ')||left('RS',2),
||left( ' ',5, ' ')||left("Direct Optimized (DO) SMB parms:",32)
bf.5 = left('obs',3) right('Job name',8) left('Run date',11),
left('Elapsed time',14) left('Cluster/Component name',30),
right('Records',9) right('ret'ved',8),
right('Excp',6) left('mode',4),
left('bias',4)  left('SMB',5),
left('used user space data index',31),
left('bit fixed',10)  right('vsp',5),
right('hwt',5)  right('b31',5),
right('cb31',5)  right('ivs',5),
right('rer',5)  right('mer',5)  right('hyp',5)

bf.6 = left('-',2,5,'-')  
"EXECIO * DISKW s64bf (STEM bf.)"

/*------------------------------------------------------------------*/
/* Part 2: read and decode SMF 64 records                           */
/*------------------------------------------------------------------*/

EXECIO * DISKR SMF64 ( STEM x. FINIS'
numeric digits 10
do i = 1   to x.0

/*------------------------------------------------------------------*/
/* Header/Self-defining Section                                     */
/*------------------------------------------------------------------*/

rty = c2d(substr(x.i,2,1))
if rty <> '40'x then do                    /* record type         */
  smfdate = substr(c2x(substr(x.i,7,4)),3,5) /* unpack smf date     */
  smftime = smf(c2d(substr(x.i,3,4)))        /* decode smf time     */
  term= c2d(substr(x.i,3,4))                 /* termination time    */
  jbn = substr(x.i,15,8)                    /* jobname             */
  rst = smf(c2d(substr(x.i,23,4)))           /* decode rst time     */
  init= c2d(substr(x.i,23,4))                /* initiate time       */
  rsd = substr(c2x(substr(x.i,27,4)),3,5)    /* unpack rsd date     */

/*-------------------------------------------------------------------*/
/* Situation indicator                                               */
/*-------------------------------------------------------------------*/

rin = x2b(c2x(substr(x.i,39,1)))
  z1 = substr(rin,1,1)
  z2 = substr(rin,2,1)
  z3 = substr(rin,3,1)
  z4 = substr(rin,4,1)
  z5 = substr(rin,5,1)
  z6 = substr(rin,6,1)
  z7 = substr(rin,7,1)
  if  z1 =1 & z6 =1 then sit='Close on Abend  '
  else if z1 =1 then sit='Component closed'
  else if z2 =1 then sit='Vol switched    '
  else if z3 =1 then sit='No space avail  '
  else if z4 =1 then sit='Cat or CRA rec  '
  else if z5 =1 then sit='Closed type=t   '
  else if z6 =1 then sit='Abend process   '
  else if z7 =1 then sit='Close VVDS or ICF'
else                   sit='logic error'

/*------------------------------------------------------------------*/
/* Indicator of component being processed                           */
/*------------------------------------------------------------------*/

dty = x2b(c2x(substr(x.i,40,1)))          /* dataset attributes  */
w1 = substr(dty,1,1)  /* component type */
w2 = substr(dty,2,1)  /* component type */
w3 = substr(dty,3,1)  /* file format */
w4 = substr(dty,4,1)  /* file compression */
w5 = substr(dty,5,1)  /* rls */
w6 = substr(dty,6,1)  /* rls : mmf */
w7 = substr(dty,7,1)  /* file addressibility */

select
when w1 =1 then comp = 'Data'
when w2 =1 then comp = 'Index'
end

select
when w3 =1 then form = 'Extended format'
otherwise form = 'Standard format'
end

select
when w4 =1 then com  = 'Compressed'
otherwise com  = 'Non compressed'
end

select
when w5 =1 then rls  = 'RLS in effect'
when w6 =1 then rls  = 'RLS in effect MMF disabled'
otherwise rls  = 'Non rls'
end

select
when w7 =1 then addr = 'Extended addressable ds'
otherwise addr = 'Standard addressibility'
end

dnm = strip(substr(x,i,85,44))  /* dataset name */
hlq = substr(dnm,1,3)  /* ds hlq construct */
hlqt = substr(dnm,1,11)  /* test ds hlq */
chr = c2d(substr(x,i,131,4))  /* current high rba/ci */
esl = c2d(substr(x,i,135,2))  /* extent segment length*/
#extents = esl / 26  /* no. of extents */
offset = 135 + 2 + esl
sln = c2d(substr(x,i,offset,4))  /* stat.segment length */

Then do
select ;
when sln > 280
  then do;
*/
/*  Statistics Section at OPEN Time  */
/*-------------------------------------*/
nil = c2d(substr(x.i,offset+4,4))        /* # of index levels */
nex = c2d(substr(x.i,offset+8,4))        /* # of extents */
nlr = c2d(substr(x.i,offset+12,4))       /* # of records */
nde = c2d(substr(x.i,offset+16,4))       /* # of deletes */
nin = c2d(substr(x.i,offset+20,4))       /* # of inserts */
nup = c2d(substr(x.i,offset+24,4))       /* # of updates */
nre = c2d(substr(x.i,offset+28,4))       /* # of retrieves */
nnc = c2d(substr(x.i,offset+32,4))       /* # of ci splits */
nas = c2d(substr(x.i,offset+36,4))       /* # of ca splits */
nep = c2d(substr(x.i,offset+40,4))       /* # of excep count */
*/
/* Change in Statistics from OPEN to time of EOV and CLOSE */
/*----------------------------------------------------------*/
dil = c2d(substr(x.i,offset+48,4))       /* # of index levels chg. */
dex = c2d(substr(x.i,offset+52,4))       /* # of extents chg. */
drl = c2d(substr(x.i,offset+56,4))       /* # of records chg. */
dde = c2d(substr(x.i,offset+60,4))       /* # of deleted chg. */
din = c2d(substr(x.i,offset+64,4))       /* # of insert chg. */
dup = c2d(substr(x.i,offset+68,4))       /* # of update chg. */
dre = c2d(substr(x.i,offset+72,4))       /* # of retrieve chg. */
dcs = c2d(substr(x.i,offset+76,4))       /* # of ci splits chg. */
das = c2d(substr(x.i,offset+80,4))       /* # of ca splits chg. */
dep = c2d(substr(x.i,offset+84,4))       /* # of excep chg. */
*/
/*  Dataset Characteristics Section */
/*------------------------------------*/
dbs = c2d(substr(x.i,offset+92,4))       /* physical blocksize */
dci = c2d(substr(x.i,offset+96,4))       /* control interval size */
dls = c2d(substr(x.i,offset+100,4))      /* max. logical rec length */
dkl = c2d(substr(x.i,offset+104,4))      /* key length */
ddf = substr(x.i,offset+106,8)           /* dd name */
str = c2d(substr(x.i,offset+110,4))      /* string number */
plh = c2d(substr(x.i,offset+114,4))      /* # of concurrent strings */
bnb = c2d(substr(x.i,offset+118,4))      /* # of buffers requested */
bsp = c2d(substr(x.i,offset+122,4))      /* buffer space */
bfd = c2d(substr(x.i,offset+126,4))      /* data buffers */
bf1 = c2d(substr(x.i,offset+130,4))      /* index buffers */
*/
/* First ACB MACRF flag byte */
/*------------------------------*/
mcl = x2b(c2x(substr(x.i,offset+178,1)))
acbd = substr(mcl,1,1)                /* access data via index? key_access */
acbadr = substr(mcl,2,1)               /* access without index? rba_access */
acbcnv = substr(mcl,3,1)               /* control interval processing? */
acbcnv = substr(mcl,4,1)               /* sequential processing? */
acbdir = substr(mc1,5,1) /* direct processing? */
acbin = substr(mc1,6,1) /* input/get/read? */
acbcout = substr(mc1,7,1) /* output/put/write? */
acbubf = substr(mc1,8,1) /* user buffers? */
if (acbcout = 1) & (acbin = 1) then open='inout'
else if acbcout = 1 then open='output'
else open='input'

/* Second ACB MACRF flag byte */
mc2 = x2b(c2x(substr(x.i,offset+171,1)))
acbskp = substr(mc2,4,1) /* skip sequential processing */
acblogin = substr(mc2,5,1) /* logon indicator */
acbrest = substr(mc2,6,1) /* dataset to empty state */
acbsn = substr(mc2,7,1) /* shared control blocks' */
acbaix = substr(mc2,8,1) /* path_aix */
if (acbdir = 1) & (acbskn = 1) then mode='mix'
else if acbdir = 1 then mode='dir'
else if acbskp = 1 then mode='skp'
else mode='seq'

/* Third ACB MACRF flag byte */
mc3 = x2b(c2x(substr(x.i,offset+172,1)))
acblsr = substr(mc3,2,1) /* local shared resource */
acbgsr = substr(mc3,3,1) /* global shared resource */
acbici = substr(mc3,4,1) /* improved ci processing */
acbdfr = substr(mc3,5,1) /* deferred write */
acbsis = substr(mc3,6,1) /* sequential insert strategy */
acbncfx = substr(mc3,7,1) /* fixed control blocks */
acbmde = substr(mc3,8,1) /* vsam 31 bit addressing */
select
  when acblsr = 1 then shr='lsr'
  when acbgsr = 1 then shr='gsr'
  otherwise shr='nsr'
end
select
  when acbncfx = 1 then fix='yes' /* cont. blocks & buffers */
  otherwise fix='no' /* fixed in real storage */
end
select
  when acbmde = 1 then bufa='31' /* buffer addressing mode */
  otherwise bufa='24'
end
select
  when ACBSIS = '1' then ins='SIS' /* insert strategy used */
  otherwise ins='nis'
end

/* Fourth ACB MACRF flag byte */
```c
/*-------------------------------------------------------------*/
mc4 = x2b(c2x(substr(x.i,offset+173,1)))
acbrls = substr(mc4,1,1)  /* rls processing */
acbsnp = substr(mc4,2,1)  /* snp option */
mc43  = substr(mc4,3,1)  /* reserved */
mc44  = substr(mc4,4,1)  /* reserved */
mc45  = SUBSTR(mc4,5,1)  /* reserved */
mc46  = SUBSTR(mc4,6,1)  /* reserved */
mc47  = SUBSTR(mc4,7,1)  /* reserved */
mc48  = SUBSTR(mc4,8,1)  /* reserved */
/*-------------------------------------------------------------*/
/* SMB Restrictions */
/* MACRF parameters not supported are: */
/* UBF(USER buffering), ICI(Improved Control Interval processing), */
/* GSR(Global Shared Resources), LSR(Local Shared Resources), */
/* RLS(Record Level Sharing), AIX(Alternate Index)- pre z/OS 1.3 rel.*/
/* non-extended format VSAM files. */
/*-------------------------------------------------------------*/
if acbubf = 1            then note='USER buffering'
else if  acbici = 1   then note='ICI processing'
else if  acbaix = 1   then note='Alternate Index'
else if shr ¬='nsr'  then note='NSR required'
else if acbrls =1     then note='RLS processing'
else if w3 =Ø         then note='Extended format required'
else                  note='none'
/*-------------------------------------------------------------*/
/* SMB ACCESS BIAS Information */
/*-------------------------------------------------------------*/
smb = x2b(c2x(substr(x.i,offset+174,1)))
s1 = substr(smb,1,1)                             /* accbias via jcl*/
s2 = substr(smb,2,1)                             /* accbias via smb*/
s3 = substr(smb,3,1)                             /* bias=do used   */
s4 = substr(smb,4,1)                             /* bias=so used   */
s5 = substr(smb,5,1)                             /* bias=sw used   */
s6 = substr(smb,6,1)                             /* bias=dw used   */
s7 = substr(smb,7,1)                             /* bias=co used   */
s8 = substr(smb,8,1)                             /* bias=cr used   */
/*-------------------------------------------------------------*/
/* The way of SMB invocation ? */
/*-------------------------------------------------------------*/
select
   when s1 ='1' then smb='jcl'
   when s2 ='1' then smb='sys'
   otherwise         smb='none'
end
/*-------------------------------------------------------------*/
/* Kind of SMB optimization technique used ? */
/*-------------------------------------------------------------*/
select
   when s3 ='1' then bia='do'
```
when s4='1' then bia='so'
when s5='1' then bia='sw'
when s6='1' then bia='dw'
when s7='1' then bia='co'
when s8='1' then bia='cr'
otherwise         bia='none'
end

/*------------------------------------------------------------------*/
/* SMB DO Information                                               */
/*------------------------------------------------------------------*/

rsc = x2b(c2x(substr(x,i,offset+175,1)))
vsp = substr(rsc,1,1)                     /* do with smbvsp         */
hwt = substr(rsc,2,1)                     /* do with smbhwg         */
b31 = substr(rsc,3,1)                     /* remode31=cb used       */
cb31= substr(rsc,4,1)                     /* rmode31=buff used      */
ivs = substr(rsc,5,1)                     /* do: insufficient vs    */
rer = substr(rsc,6,1)                     /* do: reduced resource   */
mer = substr(rsc,7,1)                     /* do: minimum resource   */
hyp = substr(rsc,8,1)                     /* do:some or all hyperspace buffers*/
if comp="Index"  then buf  =bfi
   else if compont="Data" then buf  =bfd
elapstm = smf(term-init)

/* File processing & attribute report */
/* File buffers & & buffering management report */

fa=right(i,3,'Ø')    right(jbn,8),
left(elapstm,14),
left(dnm,30),
right(nlr,9),
right(dre,8),
right(din,6),
right(dup,6),
right(acbadr,3),
right(acbcnv,3),
right(dls,5),
right(dkl,3),
right((nil + dil),4),
right((ncs + dcs),10),
right((nas + das),10),
right(ins,5),
right((nex + dex),5),
left('1',8)
left(dde,6),
left(form,8)
left('2',24)

"EXECIO 1 DISKW s64fa"

bff= right(i,3,'Ø')    right(jbn,8),
left(elapstm,14),
left(dnm,30),
right(nlr,9),
right(dre,8),
right(dep,6),
right(acbadr,4),
left(bia,4),
left(smb,4),
right(bno,5),
right(acbubf,5),
left(addr,8),
left(bia,4),
left(smb,4),
right(bno,5),
right(acbubf,5),
left(bia,4),
left(smb,4),
right(bia,4),
left(smb,4),
CONCLUSION

It is true that SMB may not be the answer to all application program buffering requirements. Its main purpose is to provide a system
capability for improving performance buffering options for batch application processing beyond those provided by the standard defaults. However, if you haven’t implemented System Managed Buffering yet, the recommendations in this article can be applied to any VSAM file type, and the performance improvements for batch processing will be remarkable. If your datasets are currently in EXTENDED format, or will be converted to EXTENDED in the near future, you should be able to implement System Managed Buffering, and you will be able to achieve even better performance than is available with NSR buffering. As in all other cases, one shouldn’t make significant changes in a production dataset’s allocation unless they have been thoroughly tested.

REFERENCES

*z/OS DFSMS: Using Data Sets (SC26-7410)*

*z/OS V1R3: DFSMS Technical Guide (SG24-6569-00)*

*VSAM Demystified* (September 2003 edition) (SG24-6105)

*Mile Pekic*

*Systems Programmer (Serbia)*

© Xephon 2005

**BPXMTEXT utility**

As non-Unix users, we are constantly challenged by the new USS world and make lots of stupid errors! Deciphering the resultant error messages is not really intuitive. No message numbers, eight digit error codes, etc.

But IBM has delivered a ‘goody’ to help us! SYS1.SBPXEXEC(BPXTEXT) contains an EXEC that displays short explanation messages.

For example, if you try to dismount SYS1.ROOT using ISHELL, you get the following message, which is not very clear:
Ouch! What is the meaning of the reason code 058800AA?

With BPXMTEXT, it is easy to get the answer. You only have to enter the following command on the ISPF command line:

```plaintext
BPXMTEXT 058800AA
```

And you get the answer!

```plaintext
MT       BPXMTEXT msg: 058800AA
BPXFSUMT 06/05/02
JRFsParentFs: The file system has file systems mounted on it
Action: An unmount request can be honoured only if there are no file systems mounted anywhere on the requested file system.
Use the D OMVS,FILE command from the system console to find out which file systems are mounted on the requested file system. Unmount them before retrying this request.
***

Often this is enough information to determine the real error; if not, well there is always the ‘friendly’ manual!
```

Systems Programmer (France) © Xephon 2005
WANT TO SUBSCRIBE?

Each monthly issue of MVS Update is packed with ideas for improving your MVS installation – and all of the technical details necessary for you to put those ideas into practice.

With MVS Update you get:
- A practical toolkit of ready-made enhancements, developed and tested in a working environment by MVS experts throughout the world.
- Tutorial articles on system internals.
- Performance tuning tips and measurements.
- Early user reports on new products and releases.

Plus:
- MVS Update will repay the cost of subscribing many times over!

All for a fraction of the cost of a single training course! So what are you waiting for?
- Go to http://www.xephonUSA.com/subscribe now to subscribe!
- Subscribe now and receive 25% off of a 12-month subscription*!

* using promotional code MV8X41.

WANT TO CONTRIBUTE?

MVS Update is written by technical professionals just like you with a desire to share their expert knowledge with the world.

Xephon is always seeking talented individuals to contribute articles to MVS Update – and get paid for it!

If you have insight into how to make MVS more functional, secure, reliable, user friendly, or to generally improve MVS performance, please visit http://www.xephonUSA.com/contribute for more details on how you can contribute to this definitive industry publication.