185 MVS update

February 2002

In this issue

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

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

Contributions
Articles published in *MVS Update* are paid for at the rate of £170 ($260) per 1000 words and £100 ($160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 ($80) per 100 lines. In addition, there is a flat fee of £30 ($50) per article. To find out more about contributing an article, you can download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

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

Editor
Jaime Kaminski

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

Subscriptions and back-issues
A year's subscription to *MVS Update*, comprising twelve monthly issues, costs £340.00 in the UK; $505.00 in the USA and Canada; £346.00 in Europe; £352.00 in Australasia and Japan; and £350.00 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 1998 issue, are available separately to subscribers for £29.00 ($43.50) each including postage.

Printed in England.

Monitoring active started tasks or jobs

THE PROBLEM

There is often a need to monitor started tasks or batch jobs for a specific duration. In most mission-critical production environments a set of started tasks have to be up all the time and another set of job or started tasks have to be up and running during the day time. If there is any downtime, there might be a very big impact on the business. Of course, if the job or started task abends, a warning will be sent to the operator console. However, what happens if the operator fails to notice, or a started task or job does not come up on time? If there is an automated mechanism to monitor and notify the systems programmers of an attention message on the operator console, the necessary action can be taken at the earliest possible opportunity.

A SOLUTION

The following REXX routine, which uses IOF, solves the above problem. This routine reads the input file CHECK.ACTIVE.JOB input dataset to find out the jobs or started tasks (STC) to be monitored. In this input dataset we can specify the duration of time to monitor for a specific job or STC, and we can specify to monitor only during weekdays, weekends, or all the time. In the example given below, the CICSPROD region needs to be up between 07:30 am and 18:00 pm only during weekdays. If it is not up during these periods, a notification will be sent. However, there is often a need to monitor some started tasks all the time. In this scenario, the start time and end time need to be set at 00:00. This way the REXX routine will assume it has to be monitored all the time. The sample input file has a detailed description for every column.

Two kinds of notification can be achieved with this routine if any job or STC is not up during the time specified in the input dataset. One is a highlighted attention console message to the operator. There should be a message sent to the operator, regarding who he should contact if a specific subsystem is down, or he should have written down the

detailed action to be taken. In order to achieve this the REXX routine will call an Assembler routine (WTOREXX) and pass a message such as ‘Sub system/JOB CICSPROD is not up in system. Please check up’. The Assembler routine will place the above message on the operator console with a highlighted attention message to catch the operator’s attention.

The other method of notification is to send a page to the system programmer if your environment has that facility. These instructions are commented out in the REXX routine, which can be modified as per your environment.

This REXX routine can be run in batch. Sample JCL is also provided. Whenever there is a message or page from this routine, it will record an entry with the date and time in a log dataset. This is just for future reference to find out which JOB or STC has failed or was not up at a specific time.

With the help of the JES2 automatic command, we can run this REXX routine in batch every 15 or 30 minutes depending on our requirements. A sample JES2 automatic command to run every 15 minutes would be:

```
$TAA00,I=300,'$VS,''S CHECKACT'''
```

OPERATIONAL ENVIRONMENT

Use of this program is dependent on the correct customization of IOF and the Assembler routine must have been compiled and link edited in dataset CHECK.ACTIVE.LOADLIB. The sample JCL procedure has to be in any one of the JES2 procedure libraries with a proper user ID that has privileges to access the input and log datasets. The IOF minimum release should be 7C.

CHECKACT

```
/* REXX */
parse source . . myname . . . . . myenv .
say myname myenv
/*                                                                  */
/* - - CHECKACT - Checks whether jobs are active or not.            */
```

```
/*                If not sends highlighted WTO message to Console     */
/*                                                                    */
startmsg = "Sub system/JOB "
endmsg = " is not up in System. Please check up"
if myenv ¬= "IOF" then do
  "IOF *.%"myname
  exit
end
else do
  Address "TSO"
  "execio * diskr active (stem jobs. finis"
  noofjobs = jobs.Ø
  do i = 1 to noofjobs
    ADDRESS IOF
    "M"
    " "
    iscomment = substr(jobs.i,1,1)
    if iscomment = "*" then iterate
    iofjobname = strip(substr(jobs.i,2,8))
    iofstctime = substr(jobs.i,11,5)
    iofstchh = substr(iofstctime,1,2)
    iofstcmm = substr(iofstctime,4,2)
    iofstchrs = (iofstchh * 6Ø) + iofstcmm
    iofendtime = substr(jobs.i,17,5)
    iofendhh = substr(iofendtime,1,2)
    iofendmm = substr(iofendtime,4,2)
    iofendhrs = (iofendhh * 6Ø) + iofendmm
    iofweekend = substr(jobs.i,23,3)
    iofjobrstc = substr(jobs.i,27,3)
    currenthrs= time("M")
    currentday = date("W")
    if iofjobrstc = "STC" | iofjobrstc = "JOB" then
    do
      if ( (currenthrs >= iofstchrs) & (currenthrs <= iofendhrs) ) | ,
         ( (iofstchrs = Ø) & (iofendhrs = Ø) ) then
      do
        if (iofweekend = "SAT") then
        do
          select
            when currentday = "Monday" then iterate
            when currentday = "Tuesday" then iterate
            when currentday = "Wednesday" then iterate
            when currentday = "Thursday" then iterate
            when currentday = "Friday" then iterate
            when currentday = "Sunday" then iterate
            otherwise
               say "Today is Saturday"
          end          /* for Select Command */
        end            /* For IF in Week End checking */
        if (iofweekend = "SUN") then
```

```
        do
          select
            when currentday = "Monday" then iterate
            when currentday = "Tuesday" then iterate
            when currentday = "Wednesday" then iterate
            when currentday = "Thursday" then iterate
            when currentday = "Friday" then iterate
            when currentday = "Saturday" then iterate
            otherwise
               say "Today is Sunday"
          end         /* for Select Command */
        end           /* For IF in Week End checking */
        if (iofweekend = "DAY") then
        do
          select
             when currentday = "Saturday" then iterate
             when currentday = "Sunday" then iterate
             otherwise
                say "Today is Week Day"
          end   /* for Select command  */
        end
        "pre "iofjobname
/*      "section running"   */
        "extend on"
        "TSICOPY NAME(JOBNAME ran) TO(REXX) VARNAME(JNAME jran)"
        say jname
        say jran
        if (length(jname) < 3) & (length(jran) < 16) then
        do
            msg1 = startmsg||iofjobname||endmsg
            Address "TSO"
/* This same routine can be used to send a pager message to the       */
/* system programmer concerned. Do uncomment the following three      */
/* lines to sendpage from the server where you run your paging        */
/* software. In our case we run the paging software (TELALERT) on     */
/* one of our Unix systems. We use a REXEC command to send a page     */
/* from the mainframe.                                                */
/*              host = "HOSTNAME"                                     */
/*              pager = "PAGERID"                                     */
/* "REXEC -l loginid -p password "HOST" telalertc -g "pager" -m "msg1 */
/*                                                                    */
/*          CALL THE WTOREXX ASSEMBLER ROUTINE TO GET HIGHLIGHTED     */
/*          MESSAGE ON CONSOLE                                        */
   CALL WTOREXX(msg1)           logmsg = date()||' "||time()||" "||msg1
            push logmsg
            "execio 1 diskw log"
        end
      end       /* For hrs check IF statement */
      jname = ""
      jran  = ""
```

```
    end    /* for DO loop */
  end  /* for JOB/STC check if */
 Address "TSO"
 "execio * diskw log (finis"
 "free file(log)"
 "free file(active)"
 Address IOF
 "exit"
 end /* for IOF check */
 exit
```

WTOREXX

```
         EJECT
TITLE "WRITE TO CONSOLE FROM REXX PROGRAM"
WTOREXX  CSECT
         USING WTOREXX,R12
         STM   R14,R12,12(R13)
         LR    R12,R15
         ST    R13,SAVEAREA+4
         LA    R1Ø,SAVEAREA
         ST    R1Ø,8(R13)
         LR    R13,R1Ø
         B     MAINLINE
         DC    CL1Ø"WTOREXX'
         DC    CL1Ø'&SYSDATE'
         DC    CL1Ø'&SYSTIME'
MAINLINE DS    ØH
         LR    R2,R1
         L     R3,16(R2)
         L     R8,2Ø(R2)
         L     R8,Ø(R8)
         USING EVALBLOCK,R8
         LR    R2,R3                    ARGLIST
MOVMSG   LA    R6,MSG
         L     R3,4(R2)                 LENGTH
         L     R2,Ø(R2)                 MESSAGE
         LR    R7,R3
         STH   R3,MSGL
         MVCL  R6,R2
         LA    R7,MSGL
         WTO   TEXT=(R7),DESC=1     DESCRIPTOR HIGHLIGHTS ON CONSOLE
         MVC   EVALBLOCK_EVLEN,=F'1'
         MVC   EVALBLOCK_EVDATA,=C'Ø'
* ------------------------------------------------------------ *
RETURN   DS    ØH
         L     R13,SAVEAREA+4    RESTORE R13
         LM    R14,R12,12(R13)   RESTORE R14 TO R12
         XR    R15,R15           ZERO RETURN CODE REG
```

```
         BR    R14                RETURN
* ------------------------------------------------------------- *
* 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
* ------------------------------------------------------------- *
SAVEAREA DC    18F'Ø' ADDRESSED BY REG 13
         EJECT
* ------------------------------------------------------------- *
MSGL     DS    H
MSG      DS    CL2ØØ
*
* ------------------------------------------------------------- *
*        EVALBLOCK DSECT                                        *
* ------------------------------------------------------------- *
         IRXEVALB
         END   WTOREXX
```

SAMPLE INPUT FILE FORMAT (CHECK.ACTIVE.JOB)

```
*  "*" AT FIRST COLUMN IN COMMENT
*  COL.POSTITION
*  2-9   : JOB NAME
*  11-15 : START ACTIVE TIME   NOTE TIME FORMAT ALWAYS : HH:MM (24 HRS)
*          IF IT SHOULD BE ACTIVE ALL THE TIME, SPECIFY ØØ:ØØ IN BOTH
*          START AND END TIME
*  17-21 : END ACTIVE TIME
*  23-25 : DAY - WEEK DAY, SAT - SATURDAY, SUN - SUNDAY, ALL - ALL DAY
*  27-29 : STC - FOR STARTUP TASK AND JOB - FOR NORMAL JOB
 CICSPROD Ø7:3Ø 18:ØØ DAY STC
 CA7      ØØ:ØØ ØØ:ØØ ALL STC
 OPCA     ØØ:ØØ ØØ:ØØ ALL STC
 PRODJOB1 18:ØØ 21:ØØ SAT JOB
```

JCL

The following JCL can be used to run the above REXX program in batch:

```
//PROC  CHECKACT
//STEP1     EXEC PGM=IKJEFT01
//STEPLIB   DD DSN=CHECK.ACTIVE.LOADLIB,DISP=SHR
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD DUMMY
//SYSOUT    DD DUMMY
//SYSTSPRT  DD SYSOUT=*
//ACTIVE    DD DSN=CHECK.ACTIVE.JOB,DISP=SHR
//LOG       DD DSN=CHECK.ACTIVE.LOG,DISP=SHR
//SYSTSIN   DD *
  CHECKACT                                    */
//  PEND
```

Muthukumar Kannaiyan
Systems Programmer (USA)

Dataset allocation information

The following REXX program was created to display some essential information about non-VSAM datasets in a catalog. The basic information (name, volume, space allocation, and so on) is complemented with a percentage indication of how full the dataset is, in terms of maximum possible allocatable space and, in the case of a PDS, of directory availability.

The program should be run in TSO batch. The only argument to pass is the catalog to list. If this is omitted, the master catalog and all the associated user catalogs are listed. However, this can take quite a while to run. If you do not need information about all the catalogs in your system, it is preferable to call the program with only those that are important.

The output is written to a 133-byte file, in a listing-style fashion, with a header and page numbers. The first column contains only the control character for the new page. This file should be allocated to the

DDname OUTFILE. I include an example job to run the program against a catalog. If you want to run it for more catalogs in the same job, simply create more steps, and change the DISP of the output dataset to MOD, if you want to use the same output file:

```
//INFOALC JOB MSGLEVEL=(1,1),MSGCLASS=X,CLASS=Z,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IKJEFT01
//SYSEXEC  DD DISP=SHR,DSN=pds.containing.INFOALOC
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//OUTFILE  DD DISP=SHR,DSN=my.output.dataset
//SYSTSIN  DD *
  PROFILE NOPREFIX
  INFOALOC  catalog.name
/*
//
```

The output listing is shown in Figure 1:

The columns have the following meaning:

- Dataset – the full dataset name.
- Volume – the volume name. Migrated datasets are not processed and are not recalled.
- Ty – type of file: PS (sequential) or PO (partitioned).
- Unit – this is the allocation unit considered for the following two columns. It is either TRAcks or CYLinders. Block specifications are converted to tracks (see the explanation below).
- Pri – primary allocation in units (cylinders or tracks).
- Sec – secondary allocation.
- Alloc – allocated space in units.
- Ext – number of extents.
- Maximum – the maximum possible space the dataset can have, in units. Equals the primary allocation plus 15 secondary allocations.
- Used – used space in units.
- % – percentage of space used relative to maximum. It expresses


```
Listing of Catalog  RSA.TERM86.CATALOG                                                                  Page 1
----------------------+----------------------------+----------+------------------+------------+--------
Datasetname           | Volume Ty Unit   Pri   Sec  | Alloc Ext | Maximum  Used  %  | Dmax Duse % |
----------------------+----------------------------+----------+------------------+------------+--------
RSA.FARES.RGO.SEQ       V54B17 PS TRA      1     1       1  1         16     1   6
RSA.FARES.RGO.SEQ.OLD   V54B17 PS TRA      1     1       1  1         16     1   6
RSA.GLIN.ACCESS.L8      V54E4C PS TRA    100    50     100  1        850     0   0
RSA.HNDUR.CNT.E0I8.MT   V54E61 PO CYL      1     5       1  1         76     1   1     50    1   2
RSA.HNDUR.CNT.STWTR.I77 V54E62 PO TRA   3237   200    3237  1       6237  3237  52     60   56  93  DIR
RSA.HENDUR.SIGC         V5445D PS TRA      5     5       5  1         80     5   6
RSA.HNDUR.SIGF.CCE      V54D24 PO TRA    100    25     100  1        475    30   6     60    3   5
RSA.HNDUR.SIGF.CCEBK    V5440C PO TRA      5    75      80  3       1130    44   4     60    5   8
RSA.JOBRC.SAIDALOG      V54B22 PS TRA      0     1       0  0         15     0   0
RSA.JOR.FINISH.BKP      V54A04 PO TRA    200    50     950 16        950   950 100     80   71  89  DIR SPC
RSA.LIST.CARTS.PCEL     V54B21 PS TRA      0    15       0  0        225     0   0
RSA.MDUMP.T03           V54B22 PS TRA      0    90       0  0       1350     0   0
RSA.ONL.CARTS.ALL.T1    V54E7C PS TRA     50    20      50  1        350     2   1
RSA.ONL.EXEC            V54D61 PO TRA      5    10      45  5        155    41  26     50   22  44
RSA.ONL.INP             V54E00 PS CYL      1     0       1  1          1     1 100                 SPC
RSA.ONL.JCL             V54D6E PO TRA      5    10      15  2        155     9   6     50    5  10
RSA.ONL.LIST            V54E7C PS TRA      1    15       1  1        226     1   0
```

Figure 1: Sample output

how near the dataset is of running out of space. If this percentage is higher than the threshold specified by variable perused_trsh, a warning (SPC) appears on the right of the line.

For PDS only:

- Dmax – directory blocks specified.
- Duse – directory blocks used.
- % – percentage of directory blocks used. If it is greater than dirused_trsh, a warning (DIR) appears on the right of the line.

A final word about units. Whenever a file is specified in blocks, I translate it to tracks, using the LISTDSI function returned variable SYSBLKSTRK (blocks per track) as the converting factor. If you consider the following example you will see why.

Suppose you allocate a file on a 3390 with a blocksize of 4096, specifying 30 blocks as the primary space and 15 blocks as the secondary space. For a blocksize of 4096, each track will have 12 blocks, as you can see if you run LISTDSI against the dataset. When you request 30 blocks as the primary space, you are in fact allocating three tracks, the number of tracks necessary to hold 30 blocks. This means your primary allocation will turn out to be 36 blocks, and not the 30 you requested. You can easily see this if you allocate the dataset and then go to 'Dataset Information' (type 'I' in front of it in ISPF/PDF 3.4): the 'Current allocation' is 36 blocks. The same is true for the secondary extensions: each new extension of '15 blocks' means, in fact, two new tracks, or 24 blocks.

As a result you could be mistaken in thinking that your dataset could grow up to 30 + 15*12 blocks, or 210 blocks, when in fact it can grow up to 3 + 2*12 tracks, or 27 tracks, or 324 blocks!

CONCLUSION

Allocation specification in blocks can be highly misleading. For that reason, I convert blocks to the real tracks, otherwise I could not have a reliable percentage of how full the dataset is. In the above example,

the fully allocated dataset would be 324 blocks allocated out of 210 ‘possible’, or 154% full!

INFOALOC

```
/* REXX *============================================================*/
/* INFOALOC - Allocation information for non-VSAM datasets.      */
/*            Argument: catalog to list. If omitted, the master  */
/* catalog and any user catalog found within it are listed.      */
/* This REXX creates a file under DDname 'outfile' containing    */
/* the essential basic allocation information about non-VSAM     */
/* datasets. If they are nearly full, either in space or in PDS  */
/* directory, there is a warning at the right end of the line.   */
/* Variables perused_trsh (% percentage of space used treshold)  */
/* and perdir_trsh (% percentage of directory used treshold)     */
/* should be appropriately set for this purpose.                 */
/*==================================================================*/
arg catalogo .
perdir_trsh  = 80
perused_trsh = 80
lines_per_page = 68
pag = 0
cab0a = "1————————————————————————"
cab0b = "————————————————————————"
cab0c = "————————————————————"
cab1a = " ————————————————————+-"
cab1b = "———————————————+——————+———"
cab1c = "———————+———————+———"
cab2a = " Datasetname                                    | "
cab2b = "Volume Ty Unit   Pri   Sec | Alloc Ext | Maxi"
cab2c = "mum  Used   %  | Dmax Duse   % |"
b = 0
if catalogo = "" then do
   cat.b = ""
   xx = outtrap(listct.)
   "listcat" usercatalog
   xx = outtrap(off)
   do b = 1 to listct.0
      cat.b = "cat("word(listct.b,3)")"
   end
end
else do
   cat.b = "cat("catalogo")"
end
do b1 = 0 to b
   xx = outtrap(list.)
   "listcat nonvsam" cat.b1
   xx = outtrap(off)
```

```
   call analyze_list
end
execio Ø diskw outfile "(finis"
exit
/*====================================================================*/
analyze_list:
f = Ø
do c = 1 to list.Ø
   if left(word(list.c,1),7) '' "NONVSAM" then iterate c
   file = word(list.c,3)
   if left(file,3) = "OMV" then iterate
   xx = listdsi(file directory norecall)
   if sysreason = Ø & (sysdsorg="PO" | sysdsorg="PS") then do
      f = f + 1
      tdsname.f  = sysdsname
      tvolume.f  = sysvolume
      tdsorg.f   = sysdsorg
      tunits.f   = sysunits
      tprimary.f = sysprimary
      tseconds.f = sysseconds
      talloc.f   = sysalloc
      tused.f    = sysused
      textents.f = sysextents
      tblkstrk.f = sysblkstrk
      tadirblk.f = sysadirblk
      tudirblk.f = sysudirblk
      if tunits.f = "BLOCK" then do
         tracks = tprimary.f / tblkstrk.f
         parse var tracks trk1 "." trk2
         if trk2 '' "" then tracks = trk1 + 1
         tprimary.f = tracks
         tracks = tseconds.f / tblkstrk.f
         parse var tracks trk1 "." trk2
         if trk2 '' "" then tracks = trk1 + 1
         tseconds.f = tracks
         used = tused.f / tblkstrk.f
         parse var used use1 "." use2
         if use2 '' "" then used = use1 + 1
         tused.f = used
         alloc = talloc.f / tblkstrk.f
         talloc.f = alloc
         tunits.f = "TRA"
      end
   end
end
dropbuf
call write_header
do z = 1 to f
   msg1 = ""
```

```
   msg2 = ""
   possible = tprimary.z + tseconds.z * 15
   if possible ' Ø then do
      perused = format((tused.z/possible)*1ØØ,3,Ø)
      if perused ' perused_trsh then msg2 = "SPC"
   end
   else do
      perused = Ø
   end
   if datatype(tudirblk.z,"W") then do
      if tudirblk.z ' Ø then do
         perdir = format((tudirblk.z/tadirblk.z)*1ØØ,3,Ø)
         if perdir ' perdir_trsh then msg1 = "DIR"
      end
      else do
         perdir = Ø
      end
   end
   else do
      perdir = ""
   end
   queue " "left(tdsname.z,45) ||,
            left(tvolume.z,7)  ||,
            left(tdsorg.z,3)   ||,
            left(tunits.z,3)   ||,
            right(tprimary.z,7)||,
            right(tseconds.z,6)||,
            right(talloc.z,8)  ||,
            right(textents.z,3)||,
            right(possible,11) ||,
            right(tused.z,6)   ||,
            right(perused,4)   ||,
            right(tadirblk.z,8)||,
            right(tudirblk.z,5)||,
            right(perdir,4)    ||,
            right(msg1,5)      ||,
            right(msg2,4)
   execio 1 diskw outfile
   line = line + 1
   if line ' lines_per_page then do
      call write_header
   end
end
return
write_header:
pag = pag + 1
line = 5
parse var cat.b1  "("catname")"
queue cabØa||cabØb||cabØc
```

```
queue " Listing of Catalog " left(catname,85),
        date() "   Page " pag
queue cab1a||cab1b||cab1c
queue cab2a||cab2b||cab2c
queue cab1a||cab1b||cab1c
queue ""
execio "*" diskw outfile
return
```


An easy way to code a checkpoint

INTRODUCTION

In our workshop the checkpoint/restart technique is widely implemented. I believe that in some cases it could become difficult to do without this feature when using tape media (3490E, 3590) with ever increasing storage capacity. It can be quite beneficial to work with checkpoint/restart if your batch application is:

- Critical.
- Performs I/O operations on tape media.
- Lasts a considerable length of time.

Then you can:

- Interrupt the operation (eg for a scheduled IPL).
- Correct any possible errors for the media and/or unit in question.

Moreover, I noticed that the most widespread function in batch operations happens to be the mere copying of a dataset from one type of media to another (usually to one with greater storage capacity).

With a few changes to the JCL, I can request the OS for a checkpoint at every EOV (Deferred C/R – EOV). However, I feel that this is insufficient with the storage capacity of current tape media.

Because of this I wrote CHKCOPY in Assembler. This program carries out a dataset copy (INF00/OUT00) and requests a checkpoint after a predetermined number of records (PARM='...').

When the program is launched in batch (JCL example 1) it can be interrupted at any point and can be relaunched with the restart JCL (JCL example 2) giving the last checkpoint number.

The dataset associated with the DD card CHKME gathers all the data needed to code a restart and, on the whole, the data used for the entire execution of the job;-, whereas the DD card CHKDS defines the checkpoint dataset.

As previously pointed out, the program is ideally applied in cases where a job does read/write operations to tape media, but it can easily be used with files that reside on DASD.

It goes without saying that in the event of an error, the program produces the appropriate messages. All the limitations regarding the type of datasets allowed and disposition recommended can be found in *DFSMS/MVS Checkpoint/Restart.*

These routines have been tested under OS/390 Version 2 Release 6 and OS/390 Version 2 Release 10.

SAMPLE JCL TO RUN CHKCOPY/CHECKPOINT (1)

```
//.....................JOB.............................
//** ------------------------------------------------------------- **
//STEP1    EXEC PGM=CHKCOPY,PARM='xxxxxxxxxxx'
//STEPLIB  DD   DISP=SHR,DSN=your.steplib
//SYSPRINT DD   SYSOUT=*,DCB=(LRECL=48,BLKSIZE=48,RECFM=F)
//SYSABEND DD   SYSOUT=*
//SYSOUT   DD   SYSOUT=*
//INF00    DD   DISP=SHR,DSN=your.input.dataset
//OUT00    DD   DISP=SHR,DSN=your.output.dataset
//CHKDS    DD   DISP=(MOD,CATLG,CATLG),DSN=your.checkpoint.dataset,
// UNIT=CKPT,SPACE=(CYL,(10,10))
//CHKME    DD   DISP=(MOD,CATLG,CATLG),DSN=your.info.dataset,
//   UNIT=CKPT,SPACE=(TRK,(1,1))
//LOG00    DD   SYSOUT=*
//** ------------------------------------------------------------- **
//STEP2    EXEC PGM=IDCAMS,COND=(0,NE,STEP1)
//SYSPRINT DD   DUMMY
```

```
//SYSIN    DD   *
 DELETE your.checkpoint.dataset
 DELETE your.info.dataset
/*
```

SAMPLE JCL TO RUN CHKCOPY/RESTART (2)

```
//.......................JOB.............................,
//   RESTART=(STEP1,C00000xx)    '++ last checkpoint ++'
//SYSCHK   DD   DISP=OLD,UNIT=CKPT,DSN=your.checkpoint.dataset
//** ------------------------------------------------------------- **
//STEP1    EXEC PGM=CHKCOPY,PARM='xxxxxxxxxxx'
//STEPLIB  DD   DISP=SHR,DSN=your.steplib
//SYSPRINT DD   SYSOUT=*,DCB=(LRECL=48,BLKSIZE=48,RECFM=F)
//SYSABEND DD   SYSOUT=*
//SYSOUT   DD   SYSOUT=*
//INF00    DD   DISP=SHR,DSN=your.input.dataset
//OUT00    DD   DISP=SHR,DSN=your.output.dataset
//CHKDS    DD   DISP=(MOD,KEEP,KEEP),DSN=your.checkpoint.dataset
//   UNIT=CKPT,SPACE=(CYL,(10,10))
//CHKME    DD   DISP=(MOD,KEEP,KEEP),DSN=your.info.dataset,
//   UNIT=CKPT,SPACE=(TRK,(1,1))
//LOG00    DD   SYSOUT=*
//** ------------------------------------------------------------- **
//STEP2    EXEC PGM=IDCAMS,COND=(0,NE,STEP1)
//SYSPRINT DD   DUMMY
//SYSIN    DD   *
 DELETE your.checkpoint.dataset
 DELETE your.info.dataset
/*
```

CHKCOPY CSECT

```
         PRINT   NOGEN
CHKCOPY  YREGS                                  REGISTER SYMBOLS
CHKCOPY  AMODE   31
CHKCOPY  RMODE   24
         SAVE    (14,R12),,CHKCOPY.&SYSDATE.&SYSTIME
         BALR    R12,0
         USING   *,R12
         LA      R2,L_SAVE
         ST      R13,L_SAVE+4
         ST      R2,8(R13)
         LA      R13,L_SAVE
** ------------------------------------------------------------- **
START    LR      R2,R1                          SAVE R1
         SR      R11,R11                        SW/RETURN CODE
         OPEN    (OUT00,(OUTPUT)),MODE=31
```

```
         OPEN    (LOGØØ,(OUTPUT)),MODE=31
         OPEN    (CHKME,(OUTPUT)),MODE=31
         OPEN    (INFØØ,(INPUT)),MODE=31
** ---------------------------------------------------------- **
PARM_JCL L       R2,Ø(R2)                       ADDRESS DATA AREA
         L       R3,Ø(R2)                       PARM AREA
         SRL     R3,16                          COUNT IN BINARY
         LA      R2,2(R2)                       ADDRESS PARM
         MVC     P_WORK,Ø(R2)
         SR      R2,R2
         PACK    P_CKPT,P_WORK
PARM_TST CH      R3,=X'ØØØB'                    LENGTH'S TEST
         BE      PARM_TSØ                       LENGTH OK (EQU 11)
         PUT     LOGØØ,R_WRKØ9
         L       R11,=F'1'
         B       L_EXIT
PARM_TSØ TRT     P_WORK,W_TRTB                  NUMERIC?
         BZ      PARM_TS1
         PUT     LOGØØ,R_WRK1Ø
         L       R11,=F'1'
         B       L_EXIT
PARM_TS1 CP      P_CKPT,=P'Ø'                   EQUAL Ø?
         BNZ     READ_DCB
         PUT     LOGØØ,R_WRK1Ø
         L       R11,=F'1'
         B       L_EXIT
** - READ DCB ------------------------------------------------ **
READ_DCB LA      R2,INFØØ                       R2 = DCB/MAP
         USING   IHADCB,R2
         MVC     W_FRMI,DCBRECFM                RECFM/INPUT
DCB_EX   MVC     W_LHRI,DCBLRECL
         LH      R3,W_LHRI                      R3 = DCBLRECL
         CVD     R3,R_WRKL                      R3 = LRECL
         UNPK    W_CTZN,R_WRKL
         OI      W_CTZN+11,X'FØ'
         MVC     R_WRKØ6+19(Ø5),W_CTZN+7
         TM      DCBRECFM,B'ØØØØØ1ØØ'           PRINT/A '.... .1Ø'
         BZ      DCB_GG
         TM      DCBRECFM,B'ØØØØØØ1Ø'           PRINT/A
         BO      DCB_GG
         MVC     R_WRKØ6+36(1),=C'A'
DCB_GG   TM      DCBRECFM,B'ØØØ1ØØØØ'           BLOCKED '...1'
         BZ      DCB_AA
         MVC     R_WRKØ6+35(1),=C'B'
DCB_AA   TM      DCBRECFM,B'1ØØØØØØØ'           FIXED   '1Ø..'
         BZ      DCB_BB
         TM      DCBRECFM,B'Ø1ØØØØØØ'           FIXED   ?
         BZ      DCB_HH
DCB_JJ   MVC     R_WRKØ6+34(1),=C'?'
         B       WR_LØ6
```

```
DCB_HH   MVC     R_WRKØ6+34(1),=C'F'
         B       WR_LØ6
DCB_BB   TM      DCBRECFM,B'Ø1ØØØØØØ'            VARBL   'Ø1..'
         BZ      DCB_JJ
         MVC     R_WRKØ6+34(1),=C'V'
WR_LØ6   LA      R5,OUTØØ
         CR      R5,R2                           TEST DCB = INFØØ
         BE      WR_OUT
         MVC     R_WRKØ6+5(5),=C'INFØØ'
         PUT     LOGØØ,R_WRKØ6
         LR      R4,R3                           R4 = LRECL/INFØØ
         DROP    R2                              FREE/R2
         LA      R2,OUTØØ                        R2 = DCB/MAP
         USING   IHADCB,R2
         MVC     W_FRMO,DCBRECFM                 RECFM/OUTPUT
         B       DCB_EX
** - VERIFY DCB INFØØ/OUTØØ ---------------------------------------- **
WR_OUT   MVC     R_WRKØ6+5(5),=C'OUTØØ'
         PUT     LOGØØ,R_WRKØ6
         CR      R3,R4                           RECL1 = RECL2?
         BE      WR_OUV
         PUT     LOGØØ,R_WRKØ8                   DISPLAY ERROR
         L       R11,=F'1'
         B       L_EXIT
WR_OUV   CLC     W_FRMI,W_FRMO                   RECFM1 = RECFM2?
         BE      REC_OK
         PUT     LOGØØ,R_WRK12                   DISPLAY ERROR
         L       R11,=F'1'
         B       L_EXIT
** ---------------------------------------------------------------- **
REC_OK   EQU     *
         DROP    R2                              FREE/R2
         SR      R4,R4
         SR      R5,R5
         SR      R2,R2
         STORAGE OBTAIN,LENGTH=(R3),ADDR=(R2)    I/O AREA = R2
** - JOBNAME ------------------------------------------------------- **
         EXTRACT W_TWRK,'S',FIELDS=(TIOT)
         L       R4,W_TWRK
         MVC     W_RJOB+Ø(8),Ø(R4)
         SR      R4,R4
** - I/O ----------------------------------------------------------- **
RE_WR    GET     INFØØ,(R2)
         PUT     OUTØØ,(R2)
         AP      C_GETP,=P'1'
         AP      C_GETT,=P'1'
         CP      P_CKPT,C_GETP
         BE      YES_CHK
         B       RE_WR
** - CHECKPOINT/ACTIVE --------------------------------------------- **
```

```
YES_CHK  SP      C_GETP,C_GETP
         CHKPT   CHKDS,W_RID,'S'
         LR      R5,R15
         CH      R5,=H'Ø'                        GOOD CHECKPOINT
         BE      OK_CHK
         CH      R5,=H'4'                        RESTART IN PROGRESS
         BNE     ERR_CHK
         PUT     LOGØØ,R_WRK13
         PUT     CHKME,R_WRK13
         B       OK_CHK
ERR_CHK  L       R11,=F'1'
         B       L_EXIT
OK_CHK   AP      C_CKPT,=P'1'
         LA      R4,W_TZON
         TIME    DEC,ZONE=LT
         ST      R1,W_TWRK                       R1/DATE OF DAY
         UNPK    W_TZON,W_TWRK
         MVC     W_RECD+2(2),3(R4)               YEAR AFTER 2ØØØ
         MVC     W_RECD+5(3),5(R4)               JULIAN DATE
         ST      RØ,W_TWRK                       RØ/TIME OF DAY
         MVC     W_TWRK+3(1),=X'ØF'
         UNPK    W_TZON,W_TWRK
         MVC     W_RECT+Ø(2),1(R4)               HOURS
         MVC     W_RECT+3(2),3(R4)               MINUTES
         MVC     W_RECT+6(2),5(R4)               SECONDS
         PUT     CHKME,R_WRK98
         B       RE_WR
** - EOF/INPUT ------------------------------------------------- **
END_INP  CP      C_GETT,=P'Ø'
         BNE     VER_CKP
         PUT     LOGØØ,R_WRKØ7
         L       R11,=F'1'
         B       L_EXIT
VER_CKP  CP      C_GETT,P_CKPT
         BNL     INP_OK
         PUT     LOGØØ,R_WRK11
         L       R11,=F'1'
         B       L_EXIT
INP_OK   PUT     LOGØØ,R_WRKØ2
** ------------------------------------------------------------ **
L_EXIT   UNPK    W_CTZN,P_CKPT
         OI      W_CTZN+11,X'FØ'
         MVC     R_WRKØ1+19(12),W_CTZN
         PUT     LOGØØ,R_WRKØ1
         UNPK    W_CTZN,C_GETT
         OI      W_CTZN+11,X'FØ'
         MVC     R_WRKØ3+19(12),W_CTZN
         PUT     LOGØØ,R_WRKØ3
         UNPK    W_CTZN,C_CKPT
         OI      W_CTZN+11,X'FØ'
```

```
         MVC     R_WRKØ4+19(12),W_CTZN
         PUT     LOGØØ,R_WRKØ4
         CLOSE   INFØØ,MODE=31
         CLOSE   OUTØØ,MODE=31
         CLOSE   LOGØØ,MODE=31
         CLOSE   CHKME,MODE=31
** ------------------------------------------------------------ **
         L       R13,L_SAVE+4
         C       R11,=F'Ø'
         BE      END_OK
         RETURN  (14,12),RC=2Ø
END_OK   RETURN  (14,12),RC=Ø
** ------------------------------------------------------------ **
C_CKPT   DC      PL7'Ø'
C_GETP   DC      PL7'Ø'
C_GETT   DC      PL7'Ø'
L_SAVE   DS      18F
P_CKPT   DC      PL7'Ø'
P_WORK   DC      CL11'Ø'
R_WRKL   DC      D'Ø'
R_WRKØ1  DC      CL8Ø'*'I  CKPT  RECORD  <<<<<<<<<<<          *<*'
R_WRKØ2  DC      CL8Ø'*>I  EOF   INPUT   OK                 *<*'
R_WRKØ3  DC      CL8Ø'*>I  TOTAL RECORD  <<<<<<<<<<<        *<*'
R_WRKØ4  DC      CL8Ø'*>I  TOTAL CKPT    <<<<<<<<<<<        *<*'
R_WRKØ6  DC      CL8Ø'*>I  XXXXX LRECL:  XXXXX  RECFM:  X   *<*'
R_WRKØ7  DC      CL8Ø'*>E  INPUT FILE IS EMPTY              *<*'
R_WRKØ8  DC      CL8Ø'*>E  LRECL/INPUT/OUTPUT IS WRONG      *<*'
R_WRKØ9  DC      CL8Ø'*>E  LENGHT  PARM NOT = 11 BYTE       *<*'
R_WRK1Ø  DC      CL8Ø'*>E  INVALID PARM                     *<*'
R_WRK11  DC      CL8Ø'*>E  INPUT RECORDS ' CHECKPOINT-NUMBER *<*'
R_WRK12  DC      CL8Ø'*>E  INFØØ/OUTØØ: INVALID RECFM       *<*'
R_WRK13  DC      CL8Ø'*>I  JOB RESTARTED                    *<*'
W_CTZN   DC      CL12'Ø'
W_FRMI   DS      C
W_FRMO   DS      C
W_LHRI   DC      BL2'Ø'
W_TWRK   DS      F
W_TZON   DS      D
R_WRK98  DS      ØF
         DC      H'8Ø'
         DC      CL5' JOB:'
W_RJOB   DC      CL8' '
         DC      CL6' DATE:'
W_RECD   DC      CL8'2ØAA/GGG'
         DC      CL6' TIME:'
W_RECT   DC      CL8'HH:MM:SS'
         DC      CL27' CHECKPOINT SUCCESSFUL ID: '
W_RID    DC      CL8' '
R_WRK99  DC      CL2' >'
```

```
W_TRTB   DC      256X'FF'
         ORG     W_TRTB+X'FØ'
         DC      1ØX'ØØ'
            ORG
** - DCB/AREA  ------------------------------------------------- **
INFØØ    DCB   DDNAME=INFØØ,MACRF=(GM),DSORG=PS,EODAD=END_INP
OUTØØ    DCB   DDNAME=OUTØØ,MACRF=(PM),RECFM=,LRECL=Ø,BLKSIZE=Ø,      *
               DSORG=PS
LOGØØ    DCB   DDNAME=LOGØØ,MACRF=(PM),RECFM=FB,LRECL=8Ø,             *
               BLKSIZE=8ØØØ,DSORG=PS
CHKME    DCB   DDNAME=CHKME,MACRF=(PM),RECFM=FB,LRECL=8Ø,             *
               BLKSIZE=8ØØØ,DSORG=PS
CHKDS    DCB   DDNAME=CHKDS,MACRF=(W),DSORG=PS,BLKSIZE=16382Ø
         DCBD  DSORG=PS
         END     CHKCOPY
```

Massimo Ambrosini
Systems Programmer (Italy)

Automating SMS configuration activation

INTRODUCTION

This article considers the process of activating new SMS configurations (new storage management policies) in the most efficient way. During daily MVS operations, one of the repetative tasks of the storage administrator is to change the SMS configuration on demand. In our data centre, there have been many times when we have changed our SMS configuration more than five times in a single day. These changes may result from new product installations, so new datasets with new HLQs have to be introduced to the system, or a new volume added to a storage group, etc.

Even though this is not a complex process, it is both tedious and routine. In addition, because there are many steps to pay attention to, we tend to make mistakes. Moreover, in case of a mistake, the fall-back process can also be painful. In a reasonable time, SMS control datasets and ACS routines have to be returned/recovered to their pre-activation states.

So I tried to make all these tasks less complicated by writing the procedures SMSNEW and SMSREST.

The usual activation and fall-back tasks have several steps and in each step special care has to be taken because there is much manual work. With the procedure SMSNEW, the activation process will turn out to be short, automatic, and controlled because it will prevent the user from dealing with the ISMF panels in the translation, verification, test, and activation steps, saving users much valuable time. In addition, this procedure, which is running in batch, will check the result of each step, and take necessary actions immediately should an error be made in any step. To summarize, the user will not have to be concerned about anything at all. The procedures will be in charge of almost everything.

The other procedure, SMSREST, provides you with the facility to return to the last SMS configuration or one of the previous SMS configurations quickly. It also runs in batch just like SMSNEW. The procedures presented in this article are of real value to users who are required to do this kind of task every day.

SHORT ALIASES USED FOR LIBRARIES IN THE ARTICLE

The article and code refers to the following datasets and their corresponding short aliases; shown on the left:

- BKPCDS – EXP.D310.SMS.BKPCDS (previous SMS configurations lib).
- ACS – EXP.D310.SMS.ACS (ACS routines library).
- SOURCE – EXP.D310.SMS.SOURCE (source code library).
- TEST – EXP.D310.SMS.TEST.LIBRARY(test case library).
- ISFOUT – EXP.D310.SMS.SDSF.TEMP (SDSF output dataset).
- TITLE – EXP.D310.SMS.TEST.TITLE (title library).
- COMMDS – SIS.D310.COMMDS.
- ACDS – SIS.D310.ACDS.

- SCDS – SIS.D310.SCDS.
- Spare ACDS – SIS.D310.ACDS.SPARE.
- Spare COMMDS – SIS.D310.COMMDS.SPARE.

HOW TO USE THE PROCEDURES

Both procedures are called from a system in the SMS complex (on the system console or on the command line of any SDSF panel, followed by a slash character) in the following way:

- S SMSNEW – to activate a new SMS configuration
- S SMSREST– to fall-back to a previous SMS configuration.

By using the MVS 'start' command, the procedures will run as a started task.

HOW LONG DOES ACTIVATION OR FALL-BACK TAKE?

Although the running time depends on the data centre and system activity at the time of the activation, both SMSNEW and SMSREST should not take more than three minutes to complete.

PRELIMINARY TASKS

The preliminary tasks are:

1 Once you have your own source library, make changes to all dataset names used in its members according to your naming conventions. To find out which members are in the source library, please consult the section *Summary of all datasets referenced in the article*. You have to change the ACS, SOURCE, TEST, and BKPCDS library names, SMS control dataset names, and other temporary/listing dataset names.

2 Copy the two procedures (SMSNEW and SMSREST) to one of your JES2 procedure libraries. Note that you also have to change all library and dataset names according to your naming conventions since there are many references to them in these procedures.

3 Create your own ACS library and copy your ACS routines into it. If each of your ACS routines has its own 'Filtlist', merge all these Filtlists and eliminate any redundant statements to build a unique Filtlist member called FILTLIST. Finally you should have this member along with the four ACS routines ($$DC, $$MC, $$SC, and $$SG). Do not forget to create header/trailer members and an optional $HISTORY member in this library.

Note: the separation of the 'Filtlist' statements from the ACS routines is vital to implementing the procedures.

4 The JCL SMSCNTL1 has to be executed to make tree dataset allocations. One is for the BKPCDS GDG dataset, another is for the TEST library (which will have test cases to be used in the 'Testing ACS Routines' process in the SMSNEW procedure), and the last one is for title members that will be used during the test.

The BKPCDS GDG dataset is defined with five generations in our installation. This means that we keep our last five SMS configurations including the current one, and we can restore and then use any of them where necessary. So you have to decide this number of generations and put it in the JCL SMSCNTL1.

5 An SMS configuration needs at least an ACDS dataset and an SCDS dataset, and you should also allocate a spare ACDS. Having a spare copy reduces the recovery process if SMS cannot access the original because of I/O errors. You have to place the spare on a device that every system in your complex can access. The JCL SMSCNTL5 will allocate it. On the other hand, the JCL SMSCNTL3 can be used to allocate an SCDS dataset when necessary.

ACTIVATING A NEW SMS CONFIGURATION

Assuming that you have adapted the procedures, REXX programs, and JCL to your environment, the detailed activation procedure is as follows:

1 Make the necessary changes to all class structures. These changes

involve, for example, adding a new SG class, adding new volumes to it, changing some parameters in the MC, adding a library name for a new 3590 Magstar tape library, etc. All these tasks are accomplished with ISMF panels. For example, to change Management Class definitions, the ISMF Option 3.4 (Alter a Management Class) is used.

2 Make the necessary changes to the ACS routines, using ISPF 3.4 Edit. This can involve the change on the ACS logic or adding new statements according to new classes/new storage groups, etc. The members that you make changes to are $$DC, $$MC, $$SC, $$SG, and FILTLIST. Do not make any changes to the members DATACLAS, MGMTCLAS, STORCLAS, and STRGROUP since they are built by the SMSNEW procedure automatically and each consists of the contents of the FILTLIST member in common. The translation process will use these members.

Note: a year ago we used to update four ACS routines in our installation, and each had a 'Filtlist'. So every time we updated the routines, we used to have trouble keeping all the Filtlists at the same level. The approach of keeping only one copy provides us with flexibility and manageability.

3 For documentation purposes, it would be better to keep track of all changes in the SMS configuration. For example, the following information can be useful for future reference: who has activated the configuration, who has asked for that change, what changes have been involved, and the date of the change. We are using the member $HISTORY for this in our installation. It is a cumulative member, so every SMS configuration change information is briefly entered at the end of this member.

4 (Optional) you may want SMSNEW to test your ACS routines. In this case, prepare some test conditions in the TEST library. You can create as many test members as you like since the SMSNEW procedure will pick up all test members during the 'Test of ACS routines' processing. There is no nomenclature for naming the test members so they are arbitrary. In our data centre we are using the members TEST1 and TEST2 in the TEST library. Also, in the TITLE library create the members TITLE1 and TITLE2.

Note: to accelerate the SMS activation processing, you can skip this step because you may not want to spend time on editing test members. On activation, you can define a test case by using the ISMF Option 7.4.1 (Define an ACS Test Case), then you can do the test by using the Option 7.4.3 (Test ACS Routines).

5 (Optional) issue the MVS command DISPLAY SMS from a system in the SMS complex through SDSF to see the status and names of the CDS datasets. Check that there is no inconsistency between dataset names that appear in the command output and dataset names in the REXX SMSREXX1. You can check out the PARMLIB member IGDSMS*xx* as well.

6 Use the S SMSNEW command from a system in the SMS complex to activate the new SMS configuration.

7 Check the results by looking through the job output and test case results. Please consult the following section for further information. Optionally, you can check the last translation time of the SCDS dataset and name of the ACS library from which the translation has been realized by using the ISMF Option 7.5 (Display ACS Object Information).

Take a look at the following example panel that results from this query:

```
                                ACS OBJECT DISPLAY
Command ===>

CDS Name  : SIS.D310.SCDS

ACS Rtn   Source Data Set ACS      Member    Last Trans  Last Date  Last Time
Type      Routine Translated from  Name      Userid      Translated Translated
-------   -----------------------  --------  ----------  ---------- ----------
DATACLAS  EXP.D310.SMS.ACS         DATACLAS  STCUSR      2001/08/23  16:33
MGMTCLAS  EXP.D310.SMS.ACS         MGMTCLAS  STCUSR      2001/08/23  16:33
STORCLAS  EXP.D310.SMS.ACS         STORCLAS  STCUSR      2001/08/23  16:33
STORGRP   EXP.D310.SMS.ACS         STRGROUP  STCUSR      2001/08/23  16:33
```

8 If you do not agree with the test case results (and therefore do not agree with the new SMS configuration), use the S SMSREST command from a system in the SMS complex to get back to the

point where you were before, and repeat this procedure starting from step1.

HOW TO CONTROL THE ACTIVATION PROCESS

In the procedure there are several steps and all of them work closely with one another. For this reason there is a variety of logs produced. To look at them individually, use the action character '?' in the NP column of the SMSNEW output in the SDSF held output panel.

Successful completion: you will have a return code of 0 on a successful activation processing. Note that a return code of 61 also means a successful activation. The only difference is the appearance of the LOOKAT4 output. Four LOOKAT*x* datasets in the SDSF Held Output queue will be similar to those shown below.

LOOKAT1

```
DC translation is successful.
MC translation is successful.
SC translation is successful.
SG translation is successful.
All ACS Objects have been successfully created and stored in the
specified SCDS.
```

LOOKAT2

```
Validation  is successful.
Validation of the ACS Routines against storage constructs is successful.
```

LOOKAT3

```
ACDS dataset has been set. The related message is:
------------------------------------------------------------------
IGD009I ACDS SWITCHED TO SIS.D310.ACDS

Active SMS configuration has been backed-up. The msg is:
------------------------------------------------------------------
IGD014I ACTIVE CONFIGURATION SAVED IN SIS.D310.ACDS.SPARE

New SMS configuration activated. The related message is:
------------------------------------------------------------------
IGD008I NEW CONFIGURATION ACTIVATED FROM SCDS SIS.D310.SCDS
```

LOOKAT4

```
*****************************************************************
**** SAVING THE ACS ROUTINES AND THE ACTIVE SCDS INTO BKPCDS  ****
*****************************************************************
The latest generation of the BKPCDS dataset was:
EXP.D310.SMS.BKPCDS.G0066V00
Back-up date was =  29 Jul 2001
-----------------------------------------------------------------
ACS routnes and SCDS dataset has been successfully backed-up.
(to the generation EXP.D310.SMS.BKPCDS.G0067V00.)
-----------------------------------------------------------------
The latest generation of the BKPCDS dataset is NOW:
EXP.D310.SMS.BKPCDS.G0067V00
Back-up date is =  30 Jul 2001
-----------------------------------------------------------------
```

You can optionally verify the outputs TRANSLAT and VALIDATE in the SMSNEW procedure listing to be more certain of successful activation.

As for the TEST listing in the job output, it has to produce a test result as you have expected; otherwise you may have to check your ACS routines again. You can correct the ACS routines as many times as necessary and execute the procedure SMSNEW until you get the best result from the Test operation.

Following unsuccessful activation processing, you will get a non-zero return code. Please consult the section *Return codes in the SMSNEW procedure* below for the meaning of the return code you received.

It would be necessary to look at the following procedure outputs to know more about the error you have had and take the necessary actions to solve it. Once you correct the errors, you can get back to executing the SMSNEW procedure again.

LOOKAT

The LOOKAT1 output will have a short summary of the translation process of all ACS routines. If you have a return code of 11, you can pinpoint here which ACS routine is in-error (there are four ACS routines) and SMS error message(s) that start with the initials ‘IGD’. Note that the TRANSLAT output may have to be checked to have a look at the entire ACS routine sources.

The LOOKAT2 output will have a short summary of the validation process of all ACS routines against storage constructs. In case you have a return code of 12, you can see here a message saying that the validation process is not OK and giving SMS error message(s) that start with the initials 'IGD'.

The LOOKAT3 output will include the response messages against a series of SETSMS commands issued through SMSREXX1.

The LOOKAT4 output will have informational messages with regard to saving the current SCDS dataset and its corresponding ACS routines in the BKPCDS dataset:

- TRANSLAT – this output includes the original listing dataset of the translation processing. It will include not only all source ACS routines, but also all SMS messages issued during the translation.
- VALIDATE – this output includes the original listing dataset of the validation processing. It will contain all SMS messages issued during the validation.
- ISPLOGS – this output combines all ISPF logs that are generated throughout the SMSNEW procedure. So it will contain all the ISPF-related messages.
- TSOLOGS – this output combines all SYSTSPRT datasets that are generated throughout the SMSNEW procedure. So it will include all the TSO-related messages.
- JOBLOGS – this output combines all SYSPRINT datasets that are generated throughout the SMSNEW procedure. So it will include all system utilities' SYSPRINT messages. This log will include not only all utility control statements, but also all actions taken by those control statements.

Note: should some unhandled error messages come out during the activation, these three output datasets ISPLOGS, TSOLOGS, and JOBLOGS may be the key to determining the error. For this reason, they are designed for trouble-shooting purposes and they will not be created if the activation process is successful. (This is carried out by setting proper return codes and checking them on the COND parameters of EXEC statements in the SMSNEW.)

RETURN CODES IN THE SMSNEW PROCEDURE

These are the return codes set by the REXX programs during the activation procedure:

- RC=0 – the new SMS configuration is activated successfully.
- RC=11 – this return code is issued in step Paso5 and means that one or more ACS routines' translation has ended with a non-zero return code. When you get this RC, you have to check out all IGD* messages in the TRANSLAT output.
- RC=12 – this return code is issued in step Paso7 and means that validation of ACS routines against storage constructs has ended with a non-zero return code. When you get this RC, you have to check out all IGD* messages in the VALIDATE output.

  Note: when you get an RC of 11 and 12, all the rest of the steps will be flushed (skipped) except for step Paso18, which is aimed at giving the user more informational messages regarding TSO and/or ISPF commands used in the SMSNEW procedure. So, if the SMS administrator cannot determine the cause of the error in the logs TRANSLAT, VALIDATE, or LOOKAT*x*, they will most likely figure it out by looking at the logs generated in step Paso18 (JOBLOGS, TSOLOGS, ISPLOGS).
- RC=21 – this return code is issued in step Paso11. This means that the ACDS dataset, which is about to be set, could not be found. Note that if the SMSNEW procedure is able to reply to the message that comes with respect to this condition, you will not get 21. You will have 22 instead.
- RC=22 – this return code means the same as the return code 21 does. Additionally the REXX has replied to the message IGD040D giving 'C' in the console automatically.
- RC=31 – this return code is issued in step Paso11. This means that the active ACDS dataset could not be backed up by the SAVEACDS command since the spare ACDS dataset does not exist. It should have been allocated before. If a reply is given, you would not get this, but 32.

- RC=32 – this return code means the same as the return code 31 does. Additionally the REXX has replied the message IGD040D giving ‘C’ in the console automatically.
- RC=41 – this return code is also issued in step Paso11. This means that the SCDS dataset from which a new SMS configuration will be activated does not exist. If a reply is given, you would not get this, but 42.
- RC=42 – this return code means the same as return code 41. Additionally the REXX has replied with the message IGD040D, giving ‘C’ in the console automatically.

  Note: the message IGD040D appears on the console when one of the control datasets could not be found during the execution of the REXX SMSREXX1. Because an action has to be taken against this message, the REXX replies to this WTOR message automatically.

  For example, if the spare ACDS dataset does not exist at the time of the execution of the SMSNEW procedure, the following messages are displayed:

```
IGD056I ACDS SIS.D310.ACDS.SPARE NOT FOUND
*59 IGD040D UNABLE TO COMPLETE CONFIGURATION REQUEST: SAVE CONFIGURATION
 - REPLY 'U' TO RETRY OR 'C' TO PURGE REQUEST
```

- RC=51 – this return code means that an error has occurred when calling the SDSF program within the REXX SMSREXX1.
- RC=61 – at least one SMS configuration had been activated before in the same day (via SMSNEW procedure), so that no new BKPCDS generation has been created. This RC has to be taken as a warning, since the process has completed successfully.

  Note: this return code will let you know that the other storage administrator has made some changes to the SMS configuration before you did. If for some reason there is a lack of communication between storage administrators, this return code would be beneficial.

ALGORITHM TO RETURN TO PREVIOUS SMS CONFIGURATIONS

Use the SMSREST procedure to return to one of your previous SMS configurations. There can be two different scenarios for the fall-back case.

The first is the situation that you have just activated a new SMS configuration by using the SMSNEW procedure and for some reason you want to return to your previous SMS configuration. In this case the command S SMSREST,VER=-1 is issued either from SDSF or from the console (from a system in the SMS complex).

In the second case, let us imagine a situation where you start changing the SMS class constructs with ISMF panels and ACS routines using ISPF 3.4. You have done many changes on classes or you have added/deleted a storage group, or added some new volumes into an existing storage group. And at the same time you have updated the ACS routines, adding a new dataset prefix in the 'Filtlist'. You suddenly realize that you did something wrong and you would like to return to the point where you were initially. In this case because you have not run SMSNEW, you can easily return to your current SMS configuration and ACS routines.

This is accomplished by using the command S SMSREST,VER= 0 or the command S SMSREST. The latest generation of the BKPCDS will always have the current SMS configuration.

On the other hand, if you do not get what you have expected from the new SMS configuration, you can change the ACS routines and/or class structures as many times as necessary and execute the procedure SMSNEW until you get the best result. In this case, you do not need to use the SMSREST procedure to return to your previous situation (SMS configuration) unless you are confused or you get a severe error.

However, if you are confused or lose your concentration due to too many trials or you just thought that it would be much better to start from the beginning, you use the command S SMSREST and start changing your SMS configuration all over again.

Although it is not very common, sometimes you may want to return to your older SMS configurations. Because you will have your past

(back-level) SMS configurations in the BKPCDS GDG dataset, you can go as far back as the generation limit defined for this dataset. In this case the command you have to use would be S SMSREST,VER=-x. Here -x represents the 'x' older generation.

Important note: in the case of a severe error, after activation of a new SMS configuration (by the SMSNEW procedure), it is necessary to take actions as quickly as possible and return to your previous SMS configuration. Because you will always have a previous SMS configuration in your spare ACDS (according to the SMSNEW procedure), this is easy to do.

Simply issue the following three commands so that SMS can use the previous SMS configuration. With the first command, the previous SMS configuration is brought into the SMS address space from the spare ACDS. With the second one you get back to use the ACDS dataset to hold your active SMS configuration. The last one is to set the ACDS with its original name.

```
SETSMS ACDS(SIS.D310.ACDS.SPARE)
SETSMS SAVEACDS(SIS.D310.ACDS)
SETSMS ACDS(SIS.D310.ACDS)
```

Once everything is OK, you can use the S SMSREST command to return to your previous ACS routines and previous SCDS dataset that formed the previous error-free SMS configuration.

HOW TO CONTROL THE FALL-BACK PROCESS

On successful completion of fall-back processing, you will have a return code of 0. In this case your LOOKAT*x* datasets will have contents similar to the following:

LOOKAT1

```
********************************* TOP OF DATA ************************
IDCAMS  SYSTEM SERVICES

 DELETE SIS.D310.SCDS CLUSTER PURGE
IDC0550I ENTRY (D) SIS.D310.SCDS.DATA DELETED
IDC0550I ENTRY (C) SIS.D310.SCDS DELETED
IDC0001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0
```

```
 SET MAXCC=Ø

 IMPORT                                  -
      OUTDATASET(SIS.D31Ø.SCDS)          -
      INFILE(SOURCE)                     -
      INTOEMPTY                          -
      OBJECTS(                           -
            (SIS.D31Ø.SCDS)              -
            (SIS.D31Ø.SCDS.DATA))
IDCØ6Ø4I DATA SET BEING IMPORTED WAS EXPORTED ON   Ø8/Ø8/Ø1 AT 12:44:4Ø
IDCØ5Ø8I DATA ALLOCATION STATUS FOR VOLUME DRS3Ø2 IS Ø
IDCØØØ1I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS Ø
****************************** BOTTOM OF DATA **********************
```

This output helps you verify the date of the exported SCDS dataset that you just imported.

LOOKAT2

```
Validation  is successful.
Validation of the ACS Routines against storage constructs is successful.
```

This is the summary of the SMS VALIDATION operation. Once your previous ACS routines and SCDS dataset from the BKPCDS dataset are restored, a validation operation is performed by the SMSREST procedure. This way you will verify that all constructs chosen by any restored ACS routine are defined to the SCDS dataset which has been restored. On the other hand, you can consult the listing of the original validation processing, which is provided under the name 'VALIDATE' in the SMSREST spool output.

LOOKAT3

```
Previous SMS config. recovered. The related message is:
------------------------------------------------------------------------
IGDØØ8I NEW CONFIGURATION ACTIVATED FROM SCDS SIS.D31Ø.SCDS
```

With unsuccessful fall-back processing, you will get a non-zero return code. It would be necessary to look at the procedure output LOOKAT*x*, ISPLOGS, TSOLOGS, and JOBLOGS, to know more about the error and take the necessary action to solve it. Once you correct the errors, you can execute the SMSREST procedure again.

RETURN CODES IN SMSREST

These are the return codes set by the REXX programs during the fall-back procedure:

- RC=0 – the previous SMS configuration is restored and activated successfully.
- RC=12 – this return code is issued in step Paso5 and means that validation of ACS routines against storage constructs has ended with a non-zero return code. When you get this RC, you have to check out all IGD* messages in the VALIDATE output.
- RC=41 – this return code is issued in step Paso6. The SCDS dataset, from which a previous SMS configuration will be activated, does not exist. Normally this return code should not appear since the fall-back process is always completed upon a successful SCDS import from one of the BKPCDS generations in step Paso2. However, the SMSREST procedure takes into account the possibility that somebody can delete the SCDS dataset during the execution of the SMSREST procedure. For this reason, the code to manage this scenario is indispensable.
- RC=42 – This return code means the same as the return code 41 does. Additionally the REXX has replied with the message IGD040D, giving 'C' in the console automatically.

ADDITIONAL NOTES ABOUT SMSNEW AND SMSREST

Please note that:

1 The versions of products used in this article are: DFSMS/MVS Version 1.5, OS/390 Version 2 Release 7, ISPF 4.5, SDSF 2.5, JES2 2.7, and TSO/E Version 2 Release 6.

2 As we all know, if there is nothing to do in the ACS routines, it is not necessary to translate them into the SCDS dataset. In this case, it would be enough to change the class constructs and activate the SMS configuration by using the ISMF panels, without having to use the SMSNEW procedure.

But I would recommend that you use SMSNEW whenever there

is a change on either ACS routines or class structures or both. It might seem that SMSNEW is doing an unnecessary task, but it provides you with an easy way of returning to your previous SMS configurations.

3 I have customized the translate, validate, and test REXX EXECs of the ISMF CLIST library and incorporated them into the SMSNEW procedure. They are ACBQBAO1, ACBQBAO2, and ACBQBAIA, and can be found in the library SYS1.DGTCLIB. For this reason their source code is not included in this article. For more information, please refer to the *DFSMS/MVS Version 1 Release 5 NaviQuest User's Guide*.

4 In order to issue SETSMS system commands, the SDSF interface is used in SMSREXX1 and SMSREXX2. However, it is also possible to use other environments/interfaces. For example, 'Console host environment' is available to the REXX EXECs. Using this environment, you can invoke SETSMS commands during an extended MCS console session. To use the CONSOLE environment, you must have CONSOLE command authority. Please refer to the *OS/390 TSO/E REXX Reference* manual for more information on the TSO/E CONSOLE and address console commands.

5 You can use the following command instead of the command SDSF in the in-line procedure ISSUE_PROC of SMSREXX1 when calling the SDSF environment:

```
ADDRESS LINKMVS 'Sdsf'
```

6 It is often useful to be able to look at your current SCDS dataset. You can achieve this because SMSNEW writes the current SCDS dataset, from which your current storage management configuration is activated, into the latest generation of the BKPCDS dataset, as a member called SCDS, along with the ACS routines. And it is browsable.

You can verify that the SMS configuration change is OK by browsing this SCDS member. For example, if a new data class name and a new 'filtlist' has just been introduced into the SMS configuration by the SMSNEW procedure, they have both to

appear in this member, as well as in the ACS routines members of the latest generation of the BKPCDS dataset.

7 The SMSNEW procedure works on a 'one generation/per day' basis. Thus, if you activate your SMS configuration even 10 times a day by using the SMSNEW procedure, you will use/waste just one generation for that day in the BKPCDS dataset and the latest generation will have the SMS configuration last activated on that day. This control is made by SMSREXX5, which is called in step Paso14. The reason to do so is to protect older SMS configurations (older GDG generations), which you should always be able to return to.

Note that without the control of the one generation per day rule, every SMSNEW would cause the oldest generation to be deleted, and many activations a day would cause many past SMS configurations to be deleted. This way you would lose the possibility of returning to your older generations since they would have been rolled-out.

Note: if you activate your SMS configuration more than once a day, you will have an RC of 61. However, this does not necessarily mean that the process went wrong. It will point out the situation explained above.

8 There is a delay parameter (on the SET DELAY command) used by the SDSF program in SMSREXX1. You can find the optimum value for your system by doing several tests with different time values.

9 The documentation in the source of SMSREXX2 has been kept to a minimum because it is similar to the REXX SMSREXX1, which has plenty of commentary lines.

10 You can convert the procedures SMSNEW and SMSREST into plain JCL as well. But one point should be kept in mind: when using procedures, you must be sure that the activation or fall-back process will always run. However, if these processes are performed with plain JCL, you cannot be so sure, since there may be times when all JES2 initiators are occupied because of high system

activity. So your urgent activation request or fall-back request may take longer than you wish. For this reason, I recommend maintaining the catalogued-procedure approach as is used in this article.

BASIC CONCEPTS OF SMS CONTROL DATASETS

SMS control datasets are VSAM linear datasets that contain base configuration information, SMS class, aggregate group, optical library, tape library, optical drive, and storage group definitions, and ACS routines.

SCDS (Source Control Dataset)

An SCDS dataset contains an SMS configuration, which defines storage management policy. You can have any number of SCDS datasets, each of which has a different SMS configuration. But one of them is selected to be the installation management policy and an active working copy of it is made in an ACDS dataset.

The SCDS dataset should be allocated on a device shared by all systems in the SMS complex (see the job SMSCNTL3).

You can modify the SCDS dataset from which your current storage management policy was activated, without disrupting operations, because SMS manages storage with a copy of the SMS configuration (an ACDS) rather than with the original (an SCDS).

ACDS (Active Control Dataset)

When you activate an SCDS dataset, its contents are copied to a previously-allocated ACDS. The current ACDS contains a copy of the most recently-activated SMS configuration. All systems in an SMS complex use this configuration to manage storage. Although you can define any number of SCDS datasets, only one of them can be put in the ACDS dataset.

An ACDS dataset must reside on a shared volume, accessible from all systems in the SMS complex. To ease recovery in the case of failure, the ACDS dataset should reside on a different volume from the

COMMDS dataset. Also, a spare ACDS should be allocated on a different shared volume (see the job SMSCNTL5).

You can define more than one IGDSMS*xx* member in your parmlib, each specifying a different ACDS, but you can use only one ACDS at a time.

COMMDS (Communications Dataset)

The COMMDS serves as the primary means of SMS communication among systems in the SMS complex. The active systems in an SMS complex access the COMMDS dataset for current SMS complex information.

The COMMDS dataset must reside on a shared volume accessible from all systems in the SMS complex. To ease recovery in case of failure, the COMMDS dataset should reside on a different volume from the ACDS dataset. Also, a spare COMMDS should be allocated on a different shared volume (see the job SMSCNTL7).

The COMMDS dataset contains the name of the ACDS dataset containing the currently active storage management policy, the current utilization statistics for each system-managed volume, and other system information. You can define any number of COMMDSs, but only one can be active in an SMS complex.

RECOVERING CONTROL DATASETS

Based on the procedures presented in this article, the recovery process for each control dataset is explained below.

Recovering the SCDS

In case you lose your SCDS dataset, it has to be recovered, because it is vital and consists of your current SMS configuration. Since you cannot alter an ACDS, you cannot use it as an SCDS if the SCDS dataset is lost or deleted accidentally.

For purposes of recovery, an SCDS dataset has to be treated the same as any other VSAM linear dataset. So you can recover it from your

daily or weekly back-ups. If you use DFSMShsm to manage its availability, it will relieve you from having to allocate any spares. Once you restore it, you have to apply all changes on the SCDS since it was last backed-up.

On the other hand, considering the SMSNEW procedure, the recovery process is much easier. Every time you activate a new SMS configuration using SMSNEW, you will also be backing-up the SCDS dataset from which your current storage management policy was activated, along with its corresponding ACS routines, into the BKPCDS dataset as a new generation.

So it is enough to execute the JCL SMSCNTL2 to restore the original SCDS dataset. Better yet, you can use the procedure SMSREST. Because, SMSCNTL2 requires that you make some change to it, whereas the procedure will not require you to do anything. It will be enough to issue the command S SMSREST,VER=0 from a system in the SMS complex (via SDSF or console). On completion of this command, the active SCDS will be recovered with its original name.

Recovering the ACDS

You can recover from errors that prevent access to the ACDS dataset, because now you have a spare ACDS (SIS.D310.ACDS.SPARE). In normal circumstances, this dataset keeps the previous ACDS, which used to be the active ACDS before the last SMS activation process – because the SMSNEW procedure works this manner.

But if you have permanent I/O errors that make the ACDS dataset unreadable or unwritable, we have to intervene as quickly as possible and use this spare ACDS as the active/current ACDS.

For permanent I/O errors to the ACDS dataset, the following steps have to be performed:

1 Reply ‘C’ to the messages IGD041I and IGD040D on the operator console:

```
IGDØ41I PERMANENT I/O ERROR FOR {SCDS|ACDS|COMMDS} dsname
IGDØ4ØD UNABLE TO COMPLETE CONFIGURATION REQUEST: text - REPLY 'U' TO
       RETRY OR 'C' TO PURGE REQUEST
```

2 Copy the data from the SMS address space to the spare ACDS by issuing the following command from a system in the SMS complex:

```
SETSMS SAVEACDS(SIS.D310.ACDS.SPARE)
```

3 Tell the system to use the spare ACDS by issuing the following command:

```
SETSMS ACDS(SIS.D310.ACDS.SPARE).
```

You need to issue this command on only one system. The COMMDS dataset is updated to reflect this change. As the other systems in the SMS complex access the COMMDS dataset, they automatically switch to the new ACDS dataset.

Note: you should use only the spare ACDS (SIS.D310.ACDS.SPARE) as a current ACDS for a very short period of time, because this is supposed to be a temporary solution. You eventually delete the old ACDS (SIS.D310.ACDS) that was in-error, allocate a new one by using the JCL SMSCNTL4, and issue the following commands from a system in the SMS complex:

```
SETSMS SAVEACDS(SIS.D310.ACDS)
SETSMS ACDS(SIS.D310.ACDS)
```

The first command copies the data from the SMS address space to the ACDS dataset. The second one tells the system to use your original ACDS dataset name.

Recovering the COMMDS

If you cannot access the COMMDS, you can recover from the error by using a spare COMMDS dataset. You can use the JCL SMSCNTL7 to allocate it. All permanent errors that make the COMMDS unreadable require intervention. For permanent I/O errors to the COMMDS, the messages IGD041I and IGD070D appear on an operator console.

```
IGD041I PERMANENT I/O ERROR FOR {SCDS|ACDS|COMMDS} dsname
IGD070D SMS COMMUNICATION ERROR, REPLY 'U' TO RETRY, 'C' TO CANCEL, 'S'
TO
       SUSPEND, 'T' TO TERMINATE
```

Reply ‘S’ on the operator console and issue the following command from a system in the SMS complex:

```
SETSMS COMMDS(SIS.D31Ø.COMMDS.SPARE)      ...................(X)
```

One of the following three situations results:

1 If the spare COMMDS is empty it gets formatted automatically, and SMS writes the in-storage copy of the current COMMDS into the spare COMMDS. You then need to issue the same command (X) on each of the remaining systems in the SMS complex.

2 If the spare COMMDS is not empty but describes an ACDS that is not currently active in the SMS complex, then SMS issues the message IGD076D. This message asks if you want to use the contents of the COMMDS and the ACDS to which it points:

```
IGDØ76D ACDS dsname IN COMMDS NOT ACTIVE, REPLY 'U' TO ACTIVATE OR 'C' TO IGNORE
```

Reply ‘C’ to cause SMS to replace the contents of the spare COMMDS with the in-storage copy of the current COMMDS. You then need to issue the same command (X) on each of the remaining systems in the SMS complex.

3 If the spare COMMDS is not empty, but describes the ACDS that is currently active in the SMS complex, you need to issue the same command (X) on the remaining systems in the SMS complex.

The following message that you get will confirm that active COMMDS has just been replaced with the spare one:

```
IGDØØ9I COMMDS SWITCHED TO SIS.D31Ø.COMMDS.SPARE
```

A response of ‘S’ to IGD070D is recommended when recovering from the current COMMDS because a response of ‘C’ might result in an unrecoverable error when trying to re-access the current COMMDS. When access to the current COMMDS is suspended, SMS is able to access the new COMMDS without accessing the current COMMDS and resulting in further errors.

Without a usable COMMDS, the systems in the SMS complex have no means of communication. Other systems in the SMS complex are

aware of the error, but they are unaware of the switch to a new COMMDS dataset until you inform them.

Note: you should use the spare COMMDS (SIS.D310.COMMDS.SPARE) as a current COMMDS only for a short period of time, because it is designed to be a temporary solution. You eventually delete the old COMMDS (SIS.D310.COMMDS) that was in-error and allocate a new one by using the JCL SMSCNTL6.

Then issue the following command from a system in the SMS complex and issue the same command on each of the remaining systems in the SMS complex:

```
SETSMS COMMDS(SIS.D310.COMMDS)
```

However, if you are going to continue to use the spare COMMDS dataset for a few days, you have to update the COMMDS dataset name in the IGDSMS*xx* member of SYS1.PARMLIB for each system in the SMS complex. This has to be accomplished before the first IPL after the COMMDS failure. This applies to the spare ACDS as well, because both COMMDS and ACDS dataset names are specified in the IGDSMS*xx* member.

You may want to use an alternative one, while you can access the current COMMDS dataset, because you may want to change the disk volume on which it resides. You only need to issue the same SETSMS command (X) from one system. The other systems in the SMS complex detect the change from the old COMMDS to the new COMMDS and they automatically switch to the new one.

Notes on recovering control datasets

If an SMS system goes down (a temporary condition) or has not yet been started with SMS, then it is not part of the system-managed storage environment. When you activate SMS on the system, it uses the ACDS and COMMDS that are specified in its IGDSMS*xx* member. If the COMMDS and ACDS are different from the ones used by the remainder of the SMS complex, then the system runs as a separate SMS complex.

Consequently, when you change the current ACDS dataset or

COMMDS dataset and you want to continue with new names, update the IGDSMS*xx* member of SYS1.PARMLIB for each system in the SMS complex and do not forget to reflect this change on the procedures SMSNEW and SMSREST.

Recovering the SMS address space

If the SMS address space fails, SMS automatically attempts to restart up to six times. If SMS fails to recover its address space after the sixth restart, you have two options:

1 You can end the SMS address space and then restart using the command T SMS=*xx*, where *xx* identifies IGDSMS*xx* as the SMS initialization member. If you have cross memory interactions, users' address spaces wait for the SMS address space to restart. After the sixth automatic restart, SMS displays a message requesting the user to select the next action; retry another restart or terminate the SMS address space. This relates to the SMS address space and not the user's address space.

   Note that the T SMS=*xx* command is an abbreviation for the SET SMS=*xx* command. To eliminate confusion with the SETSMS operator command, the abbreviated form T SMS=*xx* is used in the SMS publications.

2 You can allow SMS to attempt another restart by replying 'yes' to the system-generated message IGD032D:

```
IGD032D THE SMS ADDRESS SPACE HAS RESTARTED nn TIMES. REPLY 'RESTART' TO
       RESTART OR 'C' TO CANCEL.
```

Cancelling the SMS address space

You cannot cancel SMS once it is activated. You must re-IPL to cancel the SMS address space and deactivate SMS. If the system is set up with automatic activation of SMS at IPL time and you want to deactivate SMS, you need to modify the appropriate SYS1.PARMLIB members to prevent automatic activation of SMS. For this, the IGDSSIIN module name has to be removed from the SMS entry in all of the IEFSSN*xx* PARMLIB members in SMS complex.

SMS-RELATED PARMLIB MEMBERS

Here is some brief information on PARMLIB members in which SMS is involved.

IEASYS*xx*

System parameters.

IEFSSN*xx*

Parameters that identify what subsystems are to be initialized.

IGDSMS*xx*

Initialize the Storage Management Subsystem (SMS) and specify the names of the active control dataset (ACDS) and the communications dataset (COMMDS).

IEFSSN*xx*

In this member SMS is defined to MVS as a valid subsystem. This member also defines how MVS activates the SMS address space – *xx* is the two-character suffix for the SMS initialization member, IGDSMS*xx*:

```
DSLIST    SIS.D310.PARMLIB(IEFSSNXX)
 Command ===>
******************************** Top of Data ***********************
...

SUBSYS SUBNAME(SMS)  INITRTN(IGDSSIIN) INITPARM('ID=XX,PROMPT=DISPLAY')
SUBSYS SUBNAME(JES2) PRIMARY(YES) START(NO)

...
******************************** Bottom of Data ********************
```

IGDSSIIN is the subsystem initialization routine module for SMS. Adding this module name to the SMS entry permits you to have SMS automatically started at IPL. It is recommended that the SMS record is placed before the JES2 record in IEFSSN*xx* to start SMS, because SMS must be active before starting any other subsystem.

For each system in the SMS complex, you have to update the SMS entry in IEFSSN*xx* to include the IGDSSIIN module name.

The PROMPT=DISPLAY parameter requests that the contents of IGDSMSxx be displayed, but the operator cannot modify them.

But if we have the PROMPT=YES parameter, it will request that the contents of IGDSMS*xx* be displayed, so that the operator can modify the parameters in IGDSMS*xx*. For a complete description of these parameters, see the *DFSMS/MVS DFSMSdfp Storage Administration Reference*.

IEASYS*yy*

The IEASYS*yy* member of MVS PARMLIB must have the line SMS=*xx* to identify the IGDSMS*xx* member will be used during SMS initialization:

```
DSLIST    SIS.D31Ø.PARMLIB(IEASYSXX)
 Command ===>
******************************* Top of Data ******************
...
SMS=XX,                        SELECT IGDSMSXX,
SSN=XX,                        SELECT IEFSSNXX, SUBSYSTEM NAMES
...
******************************* Bottom of Data ***************
```

IGDSMS*xx*

IGDSMS*xx* contains the information that is used to initialize the SMS address space and identify the COMMDS and ACDS datasets. The datasets that you specify for the ACDS and COMMDS pair must be the same for every system that shares DASD in your SMS configuration.

The ACDS dataset name in this member has to be used also in the REXX SMSREXX1 if the SMSNEW and SMSREST procedures are implemented. IGDSMS*xx* also sets the synchronization time interval between systems. This interval represents how many seconds SMS lets elapse before it checks the COMMDS for the status of other systems.

Example

The COMMDS and ACDS are defined as shown in the following member:

```
DSLIST    SIS.P2Ø2.PARMLIB(IGDSMSXX)
 Command ===>
******************************** Top of Data *************
SMS ACDS(SIS.D31Ø.ACDS)
    COMMDS(SIS.D31Ø.COMMDS)
    INTERVAL(15)
.....................
******************************** Bottom of Data ***********
```

When you activate SMS on the system, it uses the ACDS and COMMDS datasets that are specified in its IGDSMS*xx* member. Consequently, when you change the current ACDS or COMMDS, update the IGDSMS*xx* member of SYS1.PARMLIB for each system in the SMS complex as well as the procedures SMSNEW and SMSREST.

All of the IGDSMS*xx* members of the SMS complex must point to the same ACDS. At future IPLs, the SMS configuration contained in the ACDS is activated by all systems in the SMS complex. For example:

```
SUBSYS SUBNAME(SMS)  INITRTN(IGDSSIIN) INITPARM('ID=Ø2,PROMPT=DISPLAY')
```

indicates that the ACDS specified in IGDSMS02 contains the SMS configuration to be activated at future IPLs.

DATASETS REFERENCED IN THE ARTICLE

The following list summarizes all the datasets, including libraries and its members, used in both SMSNEW and SMSREST. Your standards may differ from the naming convention used in the article. For this reason, you should change them according to your system before putting the procedures into effect.

EXP.D310.SMS.ACS

The members of this ACS library include:

- $$DC, $$MC, $$SC, and $$SG – ACS routines (without Filtlist).

- FILTLIST – filtlist used by all classes.
- $HISTORY – brief information regarding every SMS configuration change (optional).
- YDC/YMC/YSC/YSG – header records for the ACS routines.
- ZDC/ZMC/ZSC/ZSG – trailer records for the ACS routines.
- DATACLAS, MGMTCLAS, STORCLAS, STRGROUP – ACS routines (with Filtlist).

EXP.D310.SMS.SOURCE

The members of this REXX/edit-macro and JCL library are shown below:

- SMSREXX1 to SMSREXX5 – REXX programs.
- SMSMACRO – edit macro.
- SYSIN0 to SYSIN9, SYSINA-SYSINH – Sysin control statements used by the SMSNEW and SMSREST procedure.
- SMSCNTL1 – allocate preliminary datasets.
- SMSCNTL2 – recover the SCDS from back-up.
- SMSCNTL3 – allocate an SCDS.
- SMSCNTL4 – allocate an ACDS.
- SMSCNTL5 – allocate a spare ACDS.
- SMSCNTL6 – allocate a COMMDS.
- SMSCNTL7 – allocate a spare COMMDS.

SIS.PROCLIB.PRODUCTS

This is the procedure library defined in JES2. The members include:

- SMSNEW – catalogued procedure used for activation of SMS configurations

- SMSREST – catalogued procedure used for returning to previous SMS configurations.

EXP.D310.SMS.DC.LISTING

This is the listing of the translation of ACS data class routines (FBA, 133, PS).

EXP.D310.SMS.SC.LISTING

This is the listing of the translation of ACS storage class routines (FBA, 133, PS).

EXP.D310.SMS.MC.LISTING

This is the listing of the translation of ACS management class routines (FBA, 133, PS).

EXP.D310.SMS.SG.LISTING

This is the listing of the translation of ACS storage group routines (FBA, 133, PS).

EXP.D310.SMS.VALIDATE.LISTING

This is the listing of the validation of the ACS routines (FBA, 133, PS).

EXP.D310.SMS.TEST.TITLE

This provides test headings (FBA, 133, PO). The members include:

- TITLE1 and TITLE2 – these members include heading text to be used for test results.

EXP.D310.SMS.TEST.LIBRARY

This is the ACS Test library (FB, 80, PO). The members include:

- TEST1 and TEST2 – test members which have test conditions and cases inside.

EXP.D310.SMS.TEST.RESULT1

This is the ACS test result listing with the previous configuration (FBA, 133, PS).

EXP.D310.SMS.TEST.RESULT2

This is the ACS test result listing with the new configuration (FBA, 133, PS).

EXP.D310.SMS.TEMP1

This is a temporary dataset that holds the newly-updated SCDS dataset. It is used in SMSNEW (VBS, 4100, PS).

EXP.D310.SMS.TEMP2

A temporary dataset that holds newly-updated SCDS and current ACS routines. It is used in SMSNEW (VB, 4100, PO).

EXP.D310.SMS.TEMP3

A temporary dataset that holds previous versions of SCDS and previous version of ACS routines. It is used in SMSREST (VB, 4100, PO).

EXP.D310.SMS.BKPCDS

The GDG PO dataset of which each generation holds the SCDS, ACS routines, change information and Filtlist that all belong to a specific SMS configuration.

EXP.D310.SMS.SDSF.TEMP

This temporary dataset is the ISFOUT dataset which is used to capture results of the SETSMS commands that are issued when an SDSF environment is called from the REXX SMSREXX1. It is created and deleted by this REXX, so that no pre-allocation is necessary.

SMS CONTROL DATASETS

The following SMS control datasets are used:

- SIS.D310.ACDS – SMS Active Control Dataset (VSL).
- SIS.D310.SCDS – SMS source control dataset (VSL).
- SIS.D310.COMMDS – SMS communications dataset (VSL).
- SIS.D310.ACDS.SPARE – SMS spare ACDS dataset (VSL).
- SIS.D310.COMMDS.SPARE – SMS spare communications dataset (VSL).

Editor's note: In the next edition of MVS Update *we will publish the code for SMSNEW and SMSREST.*

Atalay Gul
MVS Systems Programmer
Gas Natural Informatica SA (Spain)

Batch FTP between an MVS client and NT server – project extensions

In last month's *MVS Update* we provided the code for AISGET and AISPUT, which provide a means of supporting batch FTP between MVS and NT.

After the successful implementation of these two solutions, it was decided that we could quite simply use the 'put' routine as a substitute for our current distribution method in another area.

At the moment data (complete information for each agent) is downloaded from the mainframe once a month, is transferred to disk and then sent to our agents for them to load onto their PCs. Changes that occur between the monthly 'full information' releases are also downloaded (on a more regular basis) and are similarly sent on disk. The whole process for the 'full information' takes over a week to complete and is labour intensive.

It was decided that it would be good idea if we could provide the agent information on a server, the agents could pick it up themselves

electronically, the process could be automated and the labour intensity could be minimized.

All that needed to change was a few filenames, a couple of directories, and a few dataset names in the existing EXEC and JCL for AISGET – or so we thought. In reality nothing is ever so simple.

The main problem that we encountered was concerned with the volume of data which was required to be transferred. As we attempted to resolve this problem various side effects also had to be resolved.

PROBLEM 1

Our first problem was that our test dataset contained over 1 million records, each with a length of 514, which meant that our region size of 4MB was just a little too small, at least 510MB too small. The error message that was produced looks like this:

```
IRX0005I Error running BVSPUT, line 112: Machine storage exhausted
```

The quick and easy solution was to change the 'region' parameter in the JCL to 0MB (which is unlimited or currently in OS/390 maximum 2GB of virtual storage).

PROBLEM 2

Our next problem was that the load time for the input dataset is over 17 minutes and the job run time is over 6 hours. This was using a sample input dataset; the production dataset was estimated to be at least three times bigger. An approximate estimate suggested over 18 hours of run time. Not so good, as it should fit in a window between 2:00 am and 7:00 am (a maximum of 5 hours).

The solution to this problem came from our data processing team. They suggested we should run a sort on the input dataset, split it into several input datsets, and run several jobs in parallel to each other.

However, the temporary datasets allocated by the routine had the same name and therefore the jobs could only run sequentially. Not to be outdone I quickly introduced a new parameter, 'NN', which meant we could allocate individual temporary datasets per job.

JCL

```
...
//*****************************************************************
//* CALL:
//* BVSPUT <FILEDATE-YYMMDD> <USERID> <PASSWORD> <HLQ> <FULL/UPDT> <NN>
//*****************************************************************
...
//SYSTSIN DD *
%BVSPUT Ø11230 useridØ3 passwordØ3 testhlq full 99
//*
```

REXX

```
...
/* allocate temporary file for return file */

  tmpfile2 = env || '.TMPBST'
  tmpfile2 = tmpfile2 || nn
  tmpfile2 = tmpfile2 || '.DATAF'

/* temp dsn  = TESTHLQ.TMPBSTnn.DATAF  */
...
```

This example results in a temporary dataset with the name TESTHLQ.TMPBST99.DATAF.

PROBLEM 3

The routine, per agent/PC user, sends the contents of the temporary dataset to the agents directory on the server, ‘quits’ FTP and then allocates the temporary dataset again for the next agent/PC user. When the routine quits the FTP the session should be terminated.

Because of problems in our network, after a certain number of FTP calls, a connection reset is issued and the job fails. This could be because the firewall interprets several FTPs to the same address in a short time as a hacker attack. Alternatively, the time taken to terminate the no longer used connections is simply too long. Whichever, we needed a solution to the problem of having too many connections open and we needed to avoid the ‘connection reset’.

My solution was to introduce yet another variable for the temporary datasets, produce one dataset per agent/PC user, and send them one after another in the same FTP call. The commands to be used for the

FTP call were first stored in an array and then directly transferred to the stack prior to calling FTP.

REXX

```
...
arg filedate aisuser aispass env voll nn .  /* filedate = YYMMDD     */
qqftp.Ø = Ø
k = 1
qqftp.1 = aisuser aispass
                    /* userID and password = first command in array  */
nnnn = 1               /* counter for the temp dataset allocation    */
...
writefile:

/* allocate temporary file for put */
  tmpfile2 = env || '.TMPBST'
  tmpfile2 = tmpfile2 || nn
  tmpfile2 = tmpfile2 || '.A'
  tmpfile2 = tmpfile2 || nnnn
  tmpfile2 = tmpfile2 || '.DATAF'
  "listds '"tmpfile2"'"
  if rc = Ø then do
    say " deleting " tmpfile2
    "delete '"tmpfile2"'"
  end
  tmpbstnd = "Z" || nnnn
  say tmpfile2 "fuer pc-anw =" currentpc " wird angelegt"
  "alloc da('"tmpfile2"') dd("tmpbstnd") new blksize(514) reuse ,
       lrecl(514) catalog recfm(f b) dsorg(ps) "
  if rc ¬= Ø then
 do
    say "error in alloc, rc = " rc
    exit
  end

/* ************************************* */
/* prepare put statement for FTP call       */

  rc = LISTDSI(tmpbstnd "FILE")
  dstmpbstnd = SYSDSNAME
  dstmpbstnd = "'"dstmpbstnd"'"
  putbstnd   = "put "dstmpbstnd
  putbstnd   = putbstnd bestand

...
senddata:

  if full = 'full' | full = 'FULL' then
```

```
    changedir = "cd "currentpc"/kplhost"
  else
    changedir = "cd "currentpc"/aedhost"

  changeback = "cd /daten/ftp_data/"

  k = k + 1
  qqftp.k = changeback
  k = k + 1
  qqftp.k = changedir                        /* change directory   */
  k = k + 1
  qqftp.k = putbstnd                      /* put command >> Server  */
  k = k + 1
  qqftp.k = puteti            /* put eti (control file) >> Server  */
   nnnn = nnnn + 1
...
k = k + 1
qqftp.k = "quit"                          /* last command in array  */
do r = 1 to k
  queue qqftp.r                           /* load array onto stack  */
end

"ftp ais-ablage  (     "         /* ftp to "ais-ablage"  server    */
                                 /* using stack as input           */
```

PROBLEM 4

On the first run of the new modifications we expected about 140 temporary datasets; however, at the allocation of dataset number 39 we got the following error information messages:

```
TBASLB.TMPBST99.A39.DATAF
IKJ58506I DATA SET 'TBASLB.TMPBST99.A39.DATAF' NOT ALLOCATED, TOO MANY DATASETS+
IKJ58506I USE FREE COMMAND TO FREE UNUSED DATA SETS

IKJ56220I DATA SET TBASLB.TMPBST99.A39.DATAF NOT ALLOCATED, TOO MANY DATASETS+
IKJ56220I MAXIMUM NUMBER OF DATASET ALLOCATIONS ALLOWED BY YOUR SESSION HAS BEEN REACHED,
YOU SHOULD FREE UNUSED DATASETS
error in alloc, rc =  12
```

After searching the Internet I found that a simple solution was available, namely the use of the DYNAMNBR parameter of EXEC PGM. It now looks like this:

```
//BVSPUT   EXEC TSOBATCH,DYNAMNBR=200
```

PROBLEM 5

Here we come back to the original problem. This is that the REXX routine was not written for the huge volume of data that we needed to process. Loading so much data into virtual storage results in severe main memory usage and also in extreme amounts of paging. The next step was then to modify the routine to read and write the files sequentially.

BVSPUT.JCL

```
//BVSPUT JOB ,'USERID',
//            MSGLEVEL=(1,1),MSGCLASS=X,
//            CLASS=A,REGION=ØM,
//            NOTIFY=&SYSUID,
//            USER=,GROUP=,PASSWORD=
//*
//         SET    ENV=P
//*
//* LIB: USERID.JCL.CNTL(BVSFTP)
//* DOC:
//*
//*****************************************************************
//* AUFRUF:
//* BVSPUT <FILEDATE YYMMDD> <USER> <PASS> <HLQ> <FULL/UPDT> <NN>
//*****************************************************************
//BVSPUT   EXEC TSOBATCH,DYNAMNBR=2ØØ
//SYSPROC DD DSN=SYNPU1.SYST.CLIST,DISP=SHR
//        DD DSN=SYNPU1.V2R6.CLIST,DISP=SHR
//OUTPUT  DD SYSOUT=*
//BESTND  DD DSN=&ENV.HLQ.DØ345ØØ.AØØØØ1.BESTF(Ø),
//             DISP=SHR
//SYSTSIN DD *
%BVSPUT Ø21231 useridØ3 passwordØ3 tbaslb updt 99
//*
```

BVSPUT.EXEC

```
/* REXX ************************************************************ */
/*                          ** BVSPUT  **                            */
/*                                                                   */
/*  PUT AGENT DATA FILES TO AIS SERVER                               */
/*                                                                   */
/*  Initialize                                                       */
/*  Create control file (timestamp)                                  */
/*  Loop until end of file                                           */
/*    Create temp output file (per PC user/agent )                   */
```

```
/*    Loop until change of pc-user/agent                              */
/*      Read input record                                             */
/*      Write output record                                           */
/*    Store FTP commands for temp data in array                       */
/*    Store FTP commands for control data in array                    */
/*  Transfer FTP commands to stack                                    */
/*  Call FTP using stack                                              */
/*  Check return code of FTP                                          */
/*  Delete temp datasets                                              */
/*********************************************************************/
trace o
arg filedate aisuser aispass env full nn . /* filedate = JJMMTT       */
                                           /* userID                  */
                                           /* password                */
                                           /* environment hlq         */
                                           /* full/update flag        */
                                           /* parallel run descriptor */

qqftp.Ø = Ø                                /* array for FTP commands  */
k = 1                                      /* index for array         */
qqftp.k = aisuser aispass                  /* userID  password        */
call init                                  /* initialize              */
nnnn = 1                                   /* temp dataset counter    */
i = Ø                                      /* input record counter    */
j = Ø                                      /* output record counter   */
eof = ''                                   /* end of file flag        */
endjob = ''                                /* end of file flag        */

do forever                          /* record by record through input */
  if endjob ¬= '' then leave
  if i = Ø then                     /* until end of file              */
    do
      say "#################### START PROCESSING ################"
      call firstrec                 /* first record ?..= initialize   */
    end
  else
    do
      call nextrec
    end
end

k = k + 1
qqftp.k = "quit"                         /* last FTP command          */

do r = 1 to k                       /* transfer FTP commands to stack */
  queue qqftp.r
end

/* FTP call - using stack as input - (with timecheck)                 */
disptime = time('N')
```

```
say "************ FTP **  START : " disptime " **************"
   "ftp ais-ablage    (     "      /*    ftp to "ais-ablage"  host    */
disptime = time('N')
say "************ FTP ** END : " disptime " **************"

ftpcode = rc                          /* check return code             */
ftpcode = right(ftpcode,5,'0')        /* FTP code 5 digits long        */
ftpscmd = substr(ftpcode,1,2)         /* 1st 2 digits = subcommand     */
ftprply = substr(ftpcode,3,3)         /* last 3 digits = replycode     */

say "*****************************" /* display in output listing      */
say "FTP SUBCOMMAND = " ftpscmd
say "FTP REPLY CODE = " ftprply
say "*****************************"

ftpretcd = ''                         /* interpret severity of error   */
select
       when ftpscmd = "00"  then ftpretcd = 0
       when ftprply = "110" then ftpretcd = 0
       when ftprply = "120" then ftpretcd = 0
       when ftprply = "125" then ftpretcd = 0
       when ftprply = "150" then ftpretcd = 0
       when ftprply = "200" then ftpretcd = 0
       when ftprply = "211" then ftpretcd = 0
       when ftprply = "212" then ftpretcd = 0
       when ftprply = "213" then ftpretcd = 0
       when ftprply = "214" then ftpretcd = 0
       when ftprply = "215" then ftpretcd = 0
       when ftprply = "220" then ftpretcd = 0
       when ftprply = "221" then ftpretcd = 0
       when ftprply = "226" then ftpretcd = 0
       when ftprply = "230" then ftpretcd = 0
       when ftprply = "250" then ftpretcd = 0
       when ftprply = "257" then ftpretcd = 0
       when ftprply = "331" then ftpretcd = 0
       when ftprply = "332" then ftpretcd = 0
       when ftprply = "550" then ftpretcd = 4
  otherwise ftpretcd = 8
end
if ftpretcd > 4 then
  exit ftpretcd                       /* exit when ftp call in error   */

"delstack"                            /* clean up unused commands      */

"FREE DD(OUTPUT)"
                                      /* clean up temp datasets        */
do z = 1 to ( nnnn - 1 )
  delfile  = ''
  delfile  = env || '.TMPBST'
  delfile  = delfile  || nn
```

```
  delfile  = delfile  || '.A'
  delfile  = delfile  || z
  delfile  = delfile  || '.DATAF'

  "delete '"delfile"'"
end

return rc

/*********************************************************************** */
init:
/* initialization                                                       */
  if full = 'full' | full = 'FULL' then
  do
    bestand    = "kp"filedate".dat"     /* NT name = kpjjmmtt.dat     */
    bstndeti   = "kp"filedate".eti"     /* NT name = kpjjmmtt.eti     */
  end
  else
  do
    bestand    = "ad"filedate".dat"     /* NT name = adjjmmtt.dat     */
    bstndeti   = "ad"filedate".eti"     /* NT name = adjjmmtt.eti     */
  end

/* allocate temporary file for control file (eti) */

  tmpfile1 = env || '.TMPETI'
  tmpfile1 = tmpfile1 || nn
  tmpfile1 = tmpfile1 || '.DATAF'
  "listds '"tmpfile1"'"
  if rc = 0 then do
    say "DELETEING " tmpfile1
    "delete '"tmpfile1"'"
  end
  say "CREATING" tmpfile1
  "alloc da('"tmpfile1"') dd(tempeti) new blksize(35) reuse ,
                      lrecl(35) catalog recfm(f b) dsorg(ps) "
  if rc ¬= 0 then do
    say "error in alloc, rc = " rc
    exit rc
  end

/* prepare put statement for control file (eti) */

  rc = LISTDSI(tempeti "FILE")
  dstempeti = SYSDSNAME
  dstempeti = "'"dstempeti"'"
  puteti   = "put " dstempeti
  puteti   = puteti bstndeti

/* build control statement           */
```

```
  disptime = time('N')
  disptime1 = substr(disptime,1,2)
  disptime2 = substr(disptime,4,2)
  disptime3 = substr(disptime,7,2)
  disptime  = disptime1 || disptime2 || disptime3
  filedate = date('S')
  timestmp = filedate || disptime
  eti.1 = "00001 000000000000000"
  eti.1 = eti.1 || timestmp
  "execio * diskw tempeti (STEM eti. FINIS "
return

/* ******************************************************************* */
firstrec:
  call ftpscmds                          /* display subcommand list   */
  call ftpcodes                          /* display reply code list   */
  "execio 1 diskr " bestnd             /* read first record to stack   */
  if rc = 0 then
    do
      pull bestndr                    /* pull record from stack         */
      rc = LISTDSI(bestnd "FILE")     /* set dsbestand to datset name   */
      dsbestnd = SYSDSNAME            /* from DD statement for bestnd   */
    end
  else
    do
      bestndr = ''
      eof = "eof"                          /* set end of file flag      */
    end
  if eof ¬= '' then                  /* return code 8 when file empty  */
    do
      retcode = 8
      return retcode                       /* exit with return code     */
    end
  currentpc = substr(bestndr,7,4)    /* 1st 4 digits of agent number   */
  currentpc = currentpc || '000'     /* padded to 7 digits             */
  call nextrec                       /* continue processing            */
return
/* ******************************************************************* */
nextrec:
  if eof ¬= '' then
    do                               /* exit when last record reached  */
      call createfile
      endjob = "endjob"
      return
    end
  newpc = substr(bestndr,7,4)        /* new agent number               */
  newpc = newpc || '000'             /* padded to 7 digits             */

  if newpc ¬= currentpc then         /* when agent changes             */
    do
      call createfile                /* change output file             */
```

```
    end
  if i = Ø then
    do
      call createfile               /* create first output file       */
      rc = LISTDSI(tmpbstnd "FILE")
    end
  queue bestndr                     /* put record on stack            */
  "execio 1 diskw " tmpbstnd ""     /* write record from stack        */
  j = j + 1                         /* increment output counter       */
  "execio 1 diskr " bestnd          /* read next record to stack      */
  if rc = Ø then
    do
      pull bestndr                  /* pull input record from stack   */
      i = i + 1                     /* increment input counter        */
    end
  else
    do
      bestndr = ''
      eof = "eof"                   /* set eof when no more input     */
    end
return

/* ---------------------------------------------------------------- */
createfile:

/* ------------------------------------ */
/* prepare put statement for return message */

  "execio Ø diskw " tmpbstnd " (FINIS "  /* close output file        */
  if i ¬= Ø then
  do
    rc = LISTDSI(tmpbstnd "FILE")
    dstmpbstnd = SYSDSNAME
    dstmpbstnd = "'"dstmpbstnd"'"
    putbstnd   = "put "dstmpbstnd
    putbstnd   = putbstnd bestand
    say j "RECORDS WRITTEN TO " dstmpbstnd    /* display output count */
    j = Ø
    say i "RECORDS READ FROM " dsbestnd  /* display current input cnt */
    call stackfile                      /* prepare commands for FTP  */
    "FREE FI("tmpbstnd")"               /* free allocation           */

  end

  if eof ¬= '' then return
/* ---------------------------------------------------------- */
/* allocate temporary file number "nnnn" for return messages */

  tmpfile2 = env || '.TMPBST'
  tmpfile2 = tmpfile2 || nn
  tmpfile2 = tmpfile2 || '.A'
```

```
  tmpfile2 = tmpfile2 || nnnn
  tmpfile2 = tmpfile2 || '.DATAF'
  "listds '"tmpfile2"'"
  if rc = 0 then do                 /* does datset already exist      */
    say "DELETING " tmpfile2
    "delete '"tmpfile2"'"
  end
  tmpbstnd = "Z" || nnnn
  say "ALLOCATING " tmpfile2 "FOR " currentpc
  "alloc da('"tmpfile2"') dd("tmpbstnd") new blksize(514) reuse ,
       lrecl(514) catalog recfm(f b) dsorg(ps) "
  if rc ¬= 0 then do
    say "error in alloc, rc = " rc
    exit rc
  end

/* ************************************** */
  currentpc = newpc             /* update current agent to next agent  */
  say "#############################################################"
  say "#### " currentpc " ####"

trace o
return

/* ******************************************************************** */
/* build FTP commands together in the stack for output file transfer  */

stackfile:

                                  /* select full or update path */
  if full = 'full' | full = 'FULL' then
    changedir = "cd "currentpc"/kplhost"
  else
    changedir = "cd "currentpc"/aedhost"

  changeback = "cd /daten/ftp_data/"

  k = k + 1
  qqftp.k = changeback              /* reset directory                */
  k = k + 1
  qqftp.k = changedir               /* change directory               */
  k = k + 1
  qqftp.k = putbstnd                /* put return message >> ASS-FTP  */
  k = k + 1
  qqftp.k = puteti                  /* put cntl file    >> ASS-FTP    */

  nnnn = nnnn + 1                   /* increment agent/output counter */

/* ******************************************************************** */

return
```

```
ftpcodes:

say "+-----+-------------------------------------------------------"
say "> Code> Description                                           "
say "+-----+-------------------------------------------------------"
say "> 000 > FTP subcommand contains an incorrect parameter        "
say ">     >                                                       "
say ">     > NB : A reply code of 000 is returned from the FTP client "
say ">     >      when it detects an incorrect parameter. The FTP     "
say ">     >      client in this case does not send the command to the"
say ">     >      FTP server.                                         "
say "+-----+-------------------------------------------------------"
say "> 110 > Restart marker reply                                  "
say "+-----+-------------------------------------------------------"
say "> 120 > Service ready in nnn minutes                          "
say "+-----+-------------------------------------------------------"
say "> 125 > Data connection already open; transfer starting       "
say "+-----+-------------------------------------------------------"
say "> 150 > File status OK; about to open data connection         "
say "+-----+-------------------------------------------------------"
say "> 200 > Command OK                                            "
say "+-----+-------------------------------------------------------"
say "> 202 > Command not implemented; not used on this host        "
say "+-----+-------------------------------------------------------"
say "> 208 > Unable to delete dataset because expiration date has not "
say ">     > passed                                                "
say "+-----+-------------------------------------------------------"
say "> 211 > System status, or system help reply                   "
say "+-----+-------------------------------------------------------"
say "> 212 > Directory status                                      "
say "+-----+-------------------------------------------------------"
say "> 213 > File status                                           "
say "+-----+-------------------------------------------------------"
say "> 214 > Help message                                          "
say "+-----+-------------------------------------------------------"
say "> 215 > MVS is the operating system of this server            "
say "+-----+-------------------------------------------------------"
say "> 220 > Service ready for new user                            "
say "+-----+-------------------------------------------------------"
say "> 221 > QUIT command received                                 "
say "+-----+-------------------------------------------------------"
say "> 226 > Closing data connection; requested file action successful"
say "+-----+-------------------------------------------------------"
say "> 230 > User logged on; proceed                               "
say "+-----+-------------------------------------------------------"
say "> 250 > Requested file action OK, completed                   "
say "+-----+-------------------------------------------------------"
say "> 257 > PATH NAME created                                     "
say "+-----+-------------------------------------------------------"
say "> 331 > Send password please                                  "
```

```
say "+-----+-------------------------------------------------------------"
say "> 332 > Supply minidisk password using account                       "
say "+-----+-------------------------------------------------------------"
say "> 421 > Service not available                                        "
say "+-----+-------------------------------------------------------------"
say "> 425 > Cannot open data connection                                  "
say "+-----+-------------------------------------------------------------"
say "> 426 > Connection closed; transfer ended abnormally                 "
say "+-----+-------------------------------------------------------------"
say "> 450 > Requested file action not taken; file busy                   "
say "+-----+-------------------------------------------------------------"
say "> 451 > Requested action abended; local error in processing          "
say "+-----+-------------------------------------------------------------"
say "> 452 > Requested action not taken; insufficient storage space in"
say ">     > system                                                       "
say "+-----+-------------------------------------------------------------"
say "> 500 > Syntax error; command unrecognized                           "
say "+-----+-------------------------------------------------------------"
say "> 501 > Syntax error in parameters or arguments                      "
say "+-----+-------------------------------------------------------------"
say "> 502 > Command not implemented                                      "
say "+-----+-------------------------------------------------------------"
say "> 503 > Bad sequence of commands                                     "
say "+-----+-------------------------------------------------------------"
say "> 504 > Command not implemented for that parameter                   "
say "+-----+-------------------------------------------------------------"
say "> 530 > Not logged on                                                "
say "+-----+-------------------------------------------------------------"
say "> 532 > Need account for storing files                               "
say "+-----+-------------------------------------------------------------"
say "> 550 > Requested action not taken; file not found or no access      "
say "+-----+-------------------------------------------------------------"
say "> 551 > Requested action abended; page type unknown                  "
say "+-----+-------------------------------------------------------------"
say "> 552 > Requested file action ended abnormally; exceeded storage "
say ">     > allocation                                                   "
say "+-----+-------------------------------------------------------------"
say "> 553 > Requested action not taken; file name not allowed            "
say "+-----+-------------------------------------------------------------"
say "> 554 > Transfer aborted; unsupported SQL statement                  "
say "+-----+-------------------------------------------------------------"
return

ftpscmds:

say "+-------------+--------------------------------"
say "> Code Number > Subcommand                     "
say "+-------------+--------------------------------"
say "> 1           > AMBIGUOUS                      "
say "+-------------+--------------------------------"
say "> 2           > ?                              "
```

```
say "+-------------+--------------------------------"
say "> 3           > ACCOUNT                        "
say "+-------------+--------------------------------"
say "> 4           > APPEND                         "
say "+-------------+--------------------------------"
say "> 5           > ASCII                          "
say "+-------------+--------------------------------"
say "> 6           > BINARY                         "
say "+-------------+--------------------------------"
say "> 7           > CD                             "
say "+-------------+--------------------------------"
say "> 8           > CLOSE                          "
say "+-------------+--------------------------------"
say "> 9           > TSO                            "
say "+-------------+--------------------------------"
say "> 1Ø          > OPEN                           "
say "+-------------+--------------------------------"
say "> 11          > DEBUG                          "
say "+-------------+--------------------------------"
say "> 12          > DELIMIT                        "
say "+-------------+--------------------------------"
say "> 13          > DELETE                         "
say "+-------------+--------------------------------"
say "> 14          > DIR                            "
say "+-------------+--------------------------------"
say "> 15          > EBCDIC                         "
say "+-------------+--------------------------------"
say "> 16          > GET                            "
say "+-------------+--------------------------------"
say "> 17          > HELP                           "
say "+-------------+--------------------------------"
say "> 18          > LOCSTAT                        "
say "+-------------+--------------------------------"
say "> 19          > USER                           "
say "+-------------+--------------------------------"
say "> 2Ø          > LS                             "
say "+-------------+--------------------------------"
say "> 21          > MDELETE                        "
say "+-------------+--------------------------------"
say "> 22          > MGET                           "
say "+-------------+--------------------------------"
say "> 23          > MODE                           "
say "+-------------+--------------------------------"
say "> 24          > MPUT                           "
say "+-------------+--------------------------------"
say "> 25          > NOOP                           "
say "+-------------+--------------------------------"
say "> 26          > PASS                           "
say "+-------------+--------------------------------"
say "> 27          > PUT                            "
say "+-------------+--------------------------------"
```

```
say "> 28          > PWD                          "
say "+-------------+------------------------------"
say "> 29          > QUIT                         "
say "+-------------+------------------------------"
say "> 3Ø          > QUOTE                        "
say "+-------------+------------------------------"
say "> 31          > RENAME                       "
say "+-------------+------------------------------"
say "> 32          > SENDPORT                     "
say "+-------------+------------------------------"
say "> 33          > SENDSITE                     "
say "+-------------+------------------------------"
say "> 34          > SITE                         "
say "+-------------+------------------------------"
say "> 35          > STATUS                       "
say "+-------------+------------------------------"
say "> 36          > STRUCT                       "
say "+-------------+------------------------------"
say "> 37          > SUNIQUE                      "
say "+-------------+------------------------------"
say "> 38          > SYSTEM                       "
say "+-------------+------------------------------"
say "> 39          > TRACE                        "
say "+-------------+------------------------------"
say "> 4Ø          > TYPE                         "
say "+-------------+------------------------------"
say "> 41          > LCD                          "
say "+-------------+------------------------------"
say "> 42          > LOCSITE                      "
say "+-------------+------------------------------"
say "> 43          > LPWD                         "
say "+-------------+------------------------------"
say "> 44          > MKDIR                        "
say "+-------------+------------------------------"
say "> 45          > LMKDIR                       "
say "+-------------+------------------------------"
say "> 46          > EUCKANJI                     "
say "+-------------+------------------------------"
say "> 47          > IBMKANJI                     "
say "+-------------+------------------------------"
say "> 48          > JIS78KJ                      "
say "+-------------+------------------------------"
say "> 49          > JIS83KJ                      "
say "+-------------+------------------------------"
say "> 5Ø          > SJISKANJI                    "
say "+-------------+------------------------------"
say "> 51          > CDUP                         "
say "+-------------+------------------------------"
say "> 52          > RMDIR                        "
say "+-------------+------------------------------"
say "> 53          > HANGEUL                      "
```

```
say "+-------------+-------------------------------"
say "> 54          > KSC56Ø1                       "
say "+-------------+-------------------------------"
say "> 55          > TCHINESE                      "
say "+-------------+-------------------------------"
say "> 56          > RESTART                       "
say "+-------------+-------------------------------"
say "> 99          > UNKNOWN                       "
say "+-------------+-------------------------------"
return
```

Rolf Parker
Systems Programmer (Germany)

Converting files and translating special characters

THE PROBLEM

Nowadays, source files can exist not only in a wide range of file types (PS, PDS, PDSE, HFS, as well as various proprietary formats), but also in many record formats (F, V, U) and record lengths.

Further problems arise when additional program products are used that have their own precompilers or translators (such as DB2 and CICS). These in turn are often used in combinations which complicate matters. Whereas, for example, the C compiler allows most forms of input, the precompilers or translators often place severe restrictions on file types, record formats, and record lengths.

The following table summarizes the input handled by the C compiler, the mentioned precompilers and translators:

- C compiler – RECFM = VS, V, VB, VBS, F, FB, FBS, or FS. LRECL=32760.
- DB2 precompiler – RECFM=F; LRECL=80.
- CICS translator – RECFM=F; LRECL=80, RECFM=V; LRECL=100.

A further complication is the use of 'non-English' code pages. Several special characters required to code C (or CPP) programs, for example { has a different coding in many non-English code pages from that used by the compiler. For example, { must be written as ¸ (or as the ??< trigraph) in German. IBM provides a partial solution by providing for the use of locales that allow the use of the 'correct' coding. Unfortunately, the precompilers and translators do not handle locales. Furthermore, some header files also cannot handle locales.

To cater for the problem of code page conversion, IBM provides the EDCICONV utility described in the *C/C++ User's Guide*. This utility also allows a certain (but inadequate) degree of file format conversion.

A SOLUTION

The XEDCICNV program described here extends the functionality provided by EDCICONV to handle all usual combinations of files, but is not restricted to use with C programs. For simplicity, the XEDCICNV program provides only one code page pair: 273 to 1047, ie the conversion required in Germany. If other code pages pairs are required (eg for France) the conversion table must be modified appropriately.

The following table summarizes the combinations that XEDCICNV handles:

- Input file type – HFS (text), PS, PDS (member), PDSE (member)
- Input record format – RECFM = U, V(B), F(B)
- Maximum input record length – 32760
- Output file type – PS, PDS (member), PDSE (member)
- Output record format – RECFM = U, V(B), F(B)
- Maximum output record length – 32760.

An 'intelligent' conversion is used when the input and output record formats differ (eg input V, output F). RECFM=F output is right-padded with blanks to the specified record length. Trailing blanks are removed from RECFM=V,U output; a single blank is written if the complete output record is blank.

Note:

1 Any DCB options not explicitly specified for the output file are taken from the input file.

2 For simplicity, the complete HFS file is buffered. The supplied program allocates a 1MB buffer (sufficient for 10,000 lines each of 100 bytes). If necessary, this size could be changed easily (BUFSIZE equate).

EXEC parameters:

- /NT – No Translation (= no conversion of the special characters), ie only copy the input to the output file while performing any required file record type and record length conversions.
- /T – Translation (=conversion of the special characters), ie copy the input to the output file while performing any required file record type, record length conversions and the implicit code page conversion. This is the default value.

The following return codes are used:

- 0 – OK
- 4 – OK, input file empty
- 8 – processing error, invalid record length (LRECL of the output file is too short)
- 12 – processing error, SYSUT1 (input file) could not be opened
- 16 – processing error, SYSUT2 (output file) could not be opened
- 20 – program limitation, buffer overflow for HFS file.

The following DD statements are used:

SYSUT1 – input file

SYSUT2 – output file.

Editor's note: the code for this article will be published in the next issue of MVS Update.

Systems Programmer (Germany)

NewEra Software has begun shipping IMAGE Focus 4.0 for OS/390 and z/OS image change management. It tests, monitors, and documents changes made to system images, which helps reduce the risk of system outages due to Initial Program Load (IPL) failure.

The product addresses the issues of changes resulting from the creation and maintenance of a Parallel Sysplex and its images.

New features include the Sysplex Inspector, the Dynamic Change Inspector, and Subsystem Inspectors, which, along with the Image Inspector and the Operating System Inspector, deliver a tool set designed to find out what's changed, what went wrong, and to then fix the problem.

For further information contact:
NewEra Software, 8625 Sutter Boulevard, Suite 600, Morgan Hill, CA, 95037, USA.
Tel: (408) 201 7000.
URL: www.newera.com.

* * *

William Data Systems has begun shipping Version 3.7 of its Exigence OS/390 network problem-solving tool, enabling TCP/IP and SNA network problems caused by faulty hardware or software to be recorded and analysed without recourse to batch file tracing, printing, or manual analysis.

The product captures, inspects, formats and analyses network traffic without the use of GTF (SNA) or Ctrace (TCP/IP).

There's a facility that allows traces to be browsed, either as the trace is running, or once it has been archived. Trace information is presented with error situations highlighted. The information can be viewed from a summary presentation right down to the bit level.

Exigence also allows traces to be filtered according to the perception of where the problem is occurring and there's a context sensitive knowledge base, enabling support staff to simply point at a mysterious error code in order to receive an explanation of its meaning, together with a reason for the most likely cause of the problem and suggestions for a solution.

It runs under OS/390 Communications Server and requires an IBM TCP/IP stack.

For further information contact:
William Data Systems Ltd, Arch House, 5 High St, Old Oxted, Surrey, RH8 9LN, UK.
Tel: (01883) 723 999.
URL: http://www.willdata.com.

* * *

CA has begun shipping Advantage CA-SymDump Batch 2.0 and Advantage CA-Optimizer/II 3.0, its z/OS and OS/390 fault management solutions for helping to find and correct application problems.

Both get expanded batch abend diagnostic capabilities, including a central repository that stores abend reports in a central location, to help zoom in on relevant diagnostic details and simplify analysis of information about application failures.

For further information contact:
Computer Associates International, One Computer Associates Plaza, Islandia, NY 11749, USA.
Tel: (631) 342 6000.
URL: http://www.ca.com.