In this issue

3  NOTIFYEXTENT catalog option
6  Records with duplicate fields in a file
12 A peek into SMF30 data
31 Analysing data-in-virtual statistics
38 Waiting for datasets
45 Data conversion
56 Space abend reporter
68 Extending the life of your mainframe with in-memory table management
74 MVS news

© Xephon plc 2003
Subscriptions and back-issues
A year’s subscription to MVS Update, comprising twelve monthly issues, costs £340.00 in the UK; $505.00 in the USA and Canada; £346.00 in Europe; £352.00 in Australasia and Japan; and £350.00 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 1999 issue, are available separately to subscribers for £29.00 ($43.50) each including postage.

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

INTRODUCTION
This article describes a new CATALOG function that is available with the DFSMS APAR OW54162 (DF/SMS 1F0 – UW88212/DFSMS 1G0 – UW88213).

This new function implements the new MODIFY CATALOG,NOTIFYEXTENT(xx) parameter, which notifies you about catalogs using more than xx per cent of their maximum number of extents (123).

This enhancement is designed to help reduce system outages resulting from catalogs reaching the maximum number of extents without warning. When this occurs, any catalog operations that need to extend the catalog again will fail, and this can cause outages in online systems or major applications.

The installation may now set a threshold value as a percentage of the maximum extents, and, for any catalog that exceeds that percentage, an immediate action message is sent to the console to warn that the threshold has been exceeded.

The installation can then take preventative action as necessary to prevent an unscheduled outage. The percentage of the maximum is calculated as the allocated extents divided by 123, multiplied by 100 and rounded to the nearest per cent.

IMPLEMENTATION
The enhancement consists of four parts:

- A new form of the MODIFY CATALOG command to establish the threshold for messages:

  MODIFY CATALOG,NOTIFYEXTENT(xxx)

  where xxx is a percentage number from 0 to 99. A value of zero means that normal monitoring will be suppressed, and is the default.
This setting is retained across catalog restarts, but not IPLs. So you should add this command to your automation package to activate this functionality. It is not possible to add this command in COMMDN00 because it is processed before the CATALOG address space is active.

- New message IEC361I is issued whenever either the index or data component of a catalog exceeds the current threshold:

  IEC361I CATALOG catname HAS REACHED xxx% OF THE MAXIMUM EXTENTS

  This message is issued as an immediate action (descriptor code 2) message unless the percentage value exceeds 90%, and then it is issued as a critical action message (descriptor code 11). If normal monitoring has been disabled (e.g., the current threshold is zero), the message will still be issued when any catalog exceeds 90%.

- Message IEC348I, resulting from the MODIFY CATALOG, ALLOCATED command, has been modified to display the current percentage allocation values for all catalogs that are not deleted or closed, and that have been referenced since the last IPL:

  F CATALOG, ALLOCATED
  IEC351I CATALOG ADDRESS SPACE MODIFY COMMAND ACTIVE
  IEC348I ALLOCATED CATALOGS 263
  *CAS*******************************************************************
  * FLAGS - VOLSER - USER - CATALOG NAME % *
  * Y-I-R- DLBSØ2 0001 CATALOG.ZOSR14 4 *
  * Y-I-R- SYSBA1 0001 CATALOG.MCAT.BKUPØ1 1 *
  * YSI-E- DB2KØ1 0001 CATALOG.KDB2 22 *
  * Y-I-R- SYSBA1 0001 CATALOG.BXCF 5 *
  * Y-I-R- SYSBA1 0001 CATALOG.BT50 13 *
  * Y-I-R- SYSBA1 0001 CATALOG.BPRODUIT 11 *
  * Y-I-R- SYS$A1 0001 CATALOG.BACKUP 1 *
  * Y-I-E- SYSPA2 0001 CATALOG.MCAT.PRODØ1 1 *
  *******************************************************************
  * Y/N-ALLOCATED TO CAS, S-SMS, V-VLF, I-ISC, C-CLOSED, D-DELETED, *
  * R-SHARED, A-ATL, E-ECS SHARED, K-LOCKED *
  *CAS*******************************************************************
  IEC352I CATALOG ADDRESS SPACE MODIFY COMMAND COMPLETED

- Message IEC359I, resulting from the MODIFY CATALOG, REPORT command has been modified to display the current
setting of the extent monitoring threshold, or ‘(NONE)’ if normal monitoring is disabled.

F CATALOG, REPORT
IEC351I CATALOG ADDRESS SPACE MODIFY COMMAND ACTIVE
IEC359I CATALOG REPORT OUTPUT 423
*CAS*******************************
* CATALOG COMPONENT LEVEL = HDZ11GØ *
* CATALOG ADDRESS SPACE ASN = ØØ1D *
* SERVICE TASK UPPER LIMIT = 180 *
* SERVICE TASK LOWER LIMIT = 60 *
* HIGHEST # SERVICE TASKS = 38 *
* CURRENT # SERVICE TASKS = 38 *
* MAXIMUM # OPEN CATALOGS = 1,024 *
* ALIAS TABLE AVAILABLE = YES *
* ALIAS LEVELS SPECIFIED = 1 *
* SYS% TO SYS1 CONVERSION = OFF *
* CAS MOTHER TASK = 009A392Ø *
* CAS MODIFY TASK = 009A37ØØ *
* CAS ANALYSIS TASK = 009A0E88 *
* CAS ALLOCATION TASK = 009A33D8 *
* VOLCAT HI-LEVEL QUALIFIER = SYS1 *
* NOTIFY EXTENT = 10% *
* DELETE UCAT/VVDS WARNING = ON *
* DATASET SYNTAX CHECKING = ENABLED *
*CAS*******************************

It should be understood that this support does not provide any protection or warnings for catalogs that have no secondary space allocation.

Systems Programmer (France) © Xephon 2003

Why not share your expertise and earn money at the same time? MVS Update is looking for macros, program code, REXX EXECs, CLISTs, etc, that experienced z/OS and OS/390 users have written to make their life, or the lives of their users, easier. We will publish it (after vetting by our expert panel) and send you a cheque when the article is published. Articles can be of any length and can be sent or e-mailed to Trevor Eddolls at any of the addresses shown on page 2. A free copy of our Notes for contributors is available from our Web site at www.xephon.com/nfc.

Records with duplicate fields in a file

I was asked several times by our production people for a solution to a problem they often encountered. They receive from customers files containing identical fields in several records, and what they need is a file in which those duplicate records are eliminated. The field, identified by a length and an offset, can be in any position within the record. For this elimination of duplicates to take place, the file should be sorted by field (however, this is not always the case because there can be higher-level sort fields, like date, customer number, etc).

So I developed a program that reads a file and creates an output file, eliminating contiguous records with equal field contents and writing out just the first record of each duplicate. The program needs, as parameters, the length and offset of the field to consider. The input file can be a VSAM or a fixed-length sequential. The output file is a sequential. Any sort needed for the input file should be done within the JCL, before running the program.

At the end, the program writes out a message to sysprint stating how many records were read and how many were written.

As an example, here is a piece of JCL, examining a field with length 12 and offset 154:

```
//STEP4 EXEC PGM=DUPICAT,PARM='12,154'
//INFILE DD DISP=SHR,DSN=input_file
//OUTFILE DD DISP=SHR,DSN=output_file
//SYSPRINT DD SYSOUT=*  
```

DUPLICAT SOURCE CODE

```
*====================================================================*
* DUPLICAT - Find duplicate fields in contiguous records of a file    *
* Parameters: length (max 250) and offset of field.                 *
* DDnames: Infile, Outfile, Sysprint                                *
* This program reads an input file and copies it to an output file.  *
* If more than one contiguous record has identical fields, only the *
* first record is written to the output file.                       *
*====================================================================*
```
* The input file should be sorted by field. It can be a sequential * 
* or a VSAM. The output file must be sequential. * 
*====================================================================*
&PROGRAM SETC 'DUPLICAT'
&PROGRAM AMODE 31
&PROGRAM RMODE 24
&PROGRAM CSECT
SAVE  (14,12)
LR    R12,R15
USING &PROGRAM,R12
ST    R13,SAVEA+4
LA    R11,SAVEA
ST    R11,8(R13)
LR    R13,R11
B     GETPARMS
DC    CL16' &PROGRAM 1.2'
DC    CL8'&SYSDATE'
*====================================================================*
* Separate input parameter by comma into its components, convert * 
* them to binary form and store them in fields PARM1 and PARM2.      * 
* If any of the parms do not exist, the program terminates.          * 
*====================================================================*
GETPARMS DS   ØH
LR    R2,R1               Copy parm pointer to R2.
L     R2,Ø(Ø,R2)          Load parm address
LH    R3,Ø(R2)            Load parm length in R3
OPEN  (SYSPRINT,OUTPUT)   Open sysprint (for error msgs)
LTR   R3,R3               Any parm entered?
BZ    ERRMSGØ             No, error

LR    R6,R2
AR    R6,R3               R6: point after end of parms
LA    R6,2(Ø,R6)          Skip 2 bytes of parmlength
LA    R2,2(Ø,R2)
LR    R4,R2               R4: Current char to check
LA    R11,PARM1           Area to keep parms
XR    R9,R9               Clear length counter

* LOOPARMS EQU *
CR    R4,R6               End of parms?
BNL   CONVERT             Yes, go convert the last one
CLI   Ø(R4),C',''        Comma (separator) found?
BE    CONVERT             Yes, go convert parm
LA    R9,1(Ø,R9)          Inc index (char counter)
LA    R4,1(Ø,R4)          Inc pointer (current char)
B     LOOPARMS            And continue

* CONVERT EQU *
LTR   R9,R9               Any chars in current parm?
BZ CONVERT2 No, skip pack and cvb instructions
S R9,=F'1' Sub one for ex
EX R9,EXPACK Execute pack
LA R9,1(Ø,R9) Increment again
CVB R7,PARMPACK Convert to binary into R7
ST R7,Ø(R11) And store it

* CONVERT2 EQU *
CR R4,R6 End of all parms?
BNL CHECKPRM Yes, jump ahead
AR R2,R9 Add length to base pointer
LA R2,1(Ø,R2) And skip comma
XR R9,R9 Reset length
LR R4,R2 R4: Current char
LA R11,4(Ø,R11) Point next storearea
B LOOPARMS

* CHECKPRM CLC PARM1,=F'-1' Parm1 specified?
BE ERRMSGØ No, error
CLC PARM2,=F'-1' Same for parm2
BE ERRMSGØ
CLC PARM1,=F'250' length greater than 250?
BH ERRMSG1
S R7,=F'1'

*====================================================================*
* Check what input file we have. First open it as VSAM.             *
* If Error, assume non-VSAM file. If VSAM, test ACB for ESDS.       *
* If ESDS, modify RPL accordingly.                                  *
* Output file must be a sequential.                                *
*====================================================================*

OPENACB1 EQU * Open ACB for VSAM input file
OPEN INFILEA If error, go open sequential
LTR R15,R15
BNZ OPENDCB1
TESTCB ACB=INFILEA, X
ATRB=ESDS Check if VSAM ESDS
BNE OPENDCB2 No, go open output file

ESDSFILE1 EQU *
MODCB RPL=INFILER, X
OPTCD=ADR Modify RPL for ESDS
B OPENDCB2

OPENFILE1 EQU *
OPEN (INFILED,INPUT) Open sequential input file
LTR R15,R15
BNZ ERRMSG2
MVI FILETYP1,C'S' Set flag sequential (nonvsam)
LA R2,INFILED            Address I HADCB of input file with R2
USING I HADCB, R2

* OPENDCB2 EQU *          Open output file
OPEN (OUTFILE, OUTPUT)
LTR R15,R15
BNZ ERRMSG3*

*====================================================================*
* Read and compare loop                                             *
*====================================================================**

READFILE EQU *
XR R8,R8               Record count for input file
XR R9,R9               Record count for output file
L R3,PARM2            R3 is last position of string to
A R3,PARM1            compare. Lrecl cannot be smaller

READLOOP EQU *
CLI FILETYP1,C'V'       VSAM file?
BNE READSEQ1           No, go to sequential

READVSA1 EQU *
GET RPL=INFILER        Read VSAM file
LTR R15,R15            End of file?
BNZ EXITØ
L R4,VAREA1           Get address of data in R4.
SHOWCB RPL=INFILER,
              AREA=LRECL1,
              LENGTH=4,
              FIELDS=RECLEN
L R5,LRECL1           Get record length in R5
B COMPARE              and jump to compare

READSEQ1 EQU *
GET INFILED            Read sequential (locate method)
LR R4,R1               copy address of data to R4.
LH R5,DCBLRECL         Load R5 with record length.

COMPARE EQU *
LA R8,1(Ø,R8)          Increment input record counter
CR R5,R3               Record smaller than last position?
BL ERRMSG4             Yes, error.
LR R6,R4               Add offset to record address
A R6,PARM2             First record has no previous
C R8, =F'1'            string to compare with, so skip
BE COMPARE1            Execute compare
EX R7,EXCOMPAR         Execute compare
BE COMPARE2            If strings equal, do not write

COMPARE1 EQU *
PUT OUTFILE,(R4)   Write sequential
LA R9,1(Ø,R9)     Increment output counter

COMPARE2 EQU *
EX R7,EXMOVE Move to string
B READLOOP and read next

*====================================================================*
* Send final messages, close files, and exit                        *
*====================================================================*
EXITØ EQU *
LR R0,R8
BAL R10,UNPACK
MVC ENDMUM1,OUT10
LR R0,R9
BAL R10,UNPACK
MVC ENDMUM2,OUT10
PUT SYSPRINT,ENDMSG1
PUT SYSPRINT,ENDMSG2

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

*====================================================================*
* Subroutines and work areas                                       *
*====================================================================*
EXCOMPAR EQU *
CLC Ø(Ø,R6),STRING

EXMOVE EQU *
MVC STRING,Ø(R6)

EXPACK EQU *
PACK PARMPACK,Ø(Ø,R2)

UNPACK EQU *
CVD R0,REGDECIM
UNPK OUT12,REGDECIM
BR R10

ERRMSGØ EQU *
PUT SYSPRINT,=CL8Ø' > Parameters missing'
B EXIT1
ERRMSG1 EQU *
PUT SYSPRINT,’=CL8Ø’ > Parm1 (length) exceeds limit of 250'
B EXIT1

ERRMSG2 EQU *
PUT SYSPRINT,’=CL8Ø’ > Error opening input file'
B EXIT1

ERRMSG3 EQU *
PUT SYSPRINT,’=CL8Ø’ > Error opening output file'
B EXIT1

ERRMSG4 EQU *
PUT SYSPRINT,’=CL8Ø’ > Record smaller than compare position'
PUT SYSPRINT,’=CL8Ø’ at last read record. Program terminated'
B EXIT0

* ENDMSG1 DC 'Number of records read from INFILE . . :'
ENDNUM1 DS CL10
DC CL40'

ENDMSG2 DC 'Number of records written to OUTFILE . :'
ENDNUM2 DS CL10
DC CL40'

STRING DS OC
STRING1 DS CL250

* INFILEA ACB DDNAME=INFILE
INFLER RPL ACB=INFILEA,
OPTCD=LOC,
AREA=VAREA1,
ARG=CHAVE1

* INFILED DCB DSORG=PS, MACRF=( GL),
EODAD=EXIT0,
DDNAME=INFILE

* OUTFILE DCB DSORG=PS, MACRF=( PM),
DDNAME=OUTFILE

* SYSPRINT DCB DSORG=PS, MACRF=( PM),
LRECL=80,
DDNAME=SYSPRINT

* SAVEA DS 18F
VAREA1 DS F
CHAVE1 DS F
LRECL1 DS F
FILETYP1 DC C’ V’
PARMPACK DS D
PARM1 DC F’ -1’
PARM2 DC F’ -1’
DS 0D
REGDECIM DS CL9
DS 0F
A peek into SMF30 data

Many performance problems in an application can be quickly solved once the bottlenecks are identified. Performance monitoring tools help us identify bottlenecks, but what if a site doesn’t have those expensive tools? What if a site is planning to get rid of its performance tools to cut down costs? Sometimes it is difficult to justify the investment in performance monitoring tools because many sites using them do not use a lot of the information that they provide.

Here is a small utility program that would interpret the SMF 30 records (address space-level accounting record) and produce a report that could help identify candidates for performance tuning. This program can be customized according to your requirements – such as populating a database for performance analysis or producing customized reports. We have utilized this in the past by populating an Access database. The information in the reports is limited to analysing batch workloads.

SOFTWARE DEPENDENCIES
This program works well with the SMF30 records produced with Version 2 Release 10 of OS/390.

Compile this program after concatenating the SYS.MACLIB provided with your release of OS/390. SMF type 30 layout will be
expanded using the IFASMFR macro and the program is set to produce the report using SMF records as input.

GETTING STARTED

Although SMF data given as is to this program can contain any of the SMF record types, it will utilize only the type 30 records with sub-types 4 and 5. It is therefore preferable to filter out those records using IFASMFDP, and SORT these in order of JOBNAME/STEP name so that the step level reports generated are more readable. JCL to extract the SMF 30 (subtypes 4 and 5) and subsequently SORT the SMF 30 records is given below:

```
//**** Place Job Card Here
//STEP010 EXEC PGM=IFASMFDP
//SYSPRINT DD SYSOUT=* 
//SYSDUMP DD SYSOUT=* 
//DUMPIN DD DSIP=SHR, DSN=MY.SMF.DATASET 
//DUMPOUT DD DSN=&OUTSMF, DSIP=(NEW,PASS), 
//   DCD=(LRECL=32760, RECFM=VBS), SPACE=(CYL,(50,20), RLSE) 
//SYSLN DD *, DCB=BLKSIZE=80
//INDD(DUMPIN, OPTIONS(DUMP)) 
//OUTDD(DUMPOUT, TYPE(30(4,5))) 
START(0600) 
END(2200) 
/*
//STEP020 EXEC PGM=SORT 
//SYSPRINT DD SYSOUT=* 
//SYSPRINT DD SYSOUT=* 
//SORTIN DD DSIP=(OLD,DELETE), DSN=&OUTSMF
//SORTOUT DD DSN=MY.SORTED.SMF.DATASET, DSIP=(NEW,CATLG), 
//   DCD=(LRECL=32760, RECFM=VBS), SPACE=(CYL,(50,20), RLSE) 
//SYSLN DD *, 
//   SORT FIELDS=(247,8,CH,A,215,8,CH,A,255,2,BI,A) 
/*
```

As one may want to focus on a particular set of jobs, this utility allows the user to limit the report output by using masks at the job level. The include criterion is specified on the SYSIN card and can contain the exact job name; alternatively, it can include wildcard characters such as ‘*’ to indicate 0 or more character match, or ‘?’ to indicate exactly one character match. Multiple criteria can be given in the SYSIN card. All jobs satisfying the criteria on the SYSIN cards will be included in the generated reports.
The JCL to run this utility program is shown below:

```java
// RUN      EXEC PGM=SMF3PG
// STEPLIB  DD DISP=SHR,DSN=<loadlib containing the utility program>
// SYSIN    DD *
// JOBNAME=PRD*
// SMFIN    DD DISP=SHR,DSN=MY.SORTED.SMF.DATASET
// STEPSU   DD SYSOUT=* 
// JOBSU    DD SYSOUT=* 
// STEPTIME DD SYSOUT=* 
// JOBTIME  DD SYSOUT=* 
// STEPSTOR DD SYSOUT=* 
```

The reports produced by the sample JCL show only the jobs that start with PRD (by providing the JOBNAME=PRD* in SYSIN DD card).

**REPORTS**

This utility produces five reports that can be used to analyse the address space accounting information – DD cards JOBSU, STEPSU, JOBTIME, STEPTIME, and STEPSTOR.

A report specified by the DD card JOBTIME contains the following information at the job level:

- Job name and job number.
- Job start date (format YYYY-MM-DD) and start time (format HH:MM:SS).
- Job end date (format YYYY-MM-DD) and end time (format HH:MM:SS).
- Total CPU time taken by the job (format HH:MM:SS).
- SRB time taken by the job (format HH:MM:SS).
- Total EXCP count for the job.
- WLM class and WLM service name.

A report specified by the DD card JOBSU contains similar information to JOBTIME except that total CPU time and SRB time is expressed in terms of service units. EXCP count is expressed in terms of I/O service units.
A report specified by the DD card STEPTIME contains the following information:

- Job name and job number.
- Step start date (format YYYY-MM-DD) and start time (format HH:MM:SS).
- Step end date (format YYYY-MM-DD) and end time (format HH:MM:SS).
- Step name.
- Program name.
- CPU time taken by the step (format HH:MM:SS).
- SRB time taken by the step (format HH:MM:SS).
- EXCP count for the step.

Report specified by the DD card STEPSU contains similar information to STEPTIME except that CPU time and SRB time are expressed in terms of service units. Additionally I/O service units are also reported.

A report specified by STEPSTOR contains information about memory usage at the step level. The following information is contained in the report:

- Job name and job number.
- Step name.
- Program name.
- Step start date (format YYYY-MM-DD) and start time (format HH:MM:SS).
- Step end date (format YYYY-MM-DD) and end time (format HH:MM:SS).
- User storage below the 16MB line (in KB).
- User storage above the 16MB line (in KB).
• System storage below the 16MB line (in KB).
• System storage above the 16MB line (in KB).

The output format generated by this utility is easily portable to a spreadsheet or a database like Microsoft Access. Once the data is imported to an office tool like Excel or Access, various in-built functions (like ORDERING or SORTING, Data/Time functions) can be used to quickly get to valuable information like identifying the top consumers of CPU time, identifying long-running jobs, charting the trends over a period of time, or capacity planning. With good job naming conventions, the information can also be used for billing or charge-back purposes. Looking at resource usage in terms of service units may also be useful when moving jobs from one system to other. The STEPSTOR report can be used to see whether the program is using too much storage below the line. If yes, it can be re-compiled with DATA (31) option (if COBOL) to relieve storage below the line. In heavily paging environments, one may want to look at long-running programs consuming high storage to see whether there is potential to reduce their storage requirements. On the other hand, if the paging activity is negligible (or UIC stays at 255), one may want to look at exploiting the storage to reduce elapsed/CPU time for the programs showing high EXCP counts. Depending on what those programs do, it may be possible to use various buffering techniques like batch LSR or System Managed Buffering (SMB).

SMF30

SMF30PG TITLE 'A SAMPLE SMF30 REPORT WRITER PROGRAM'
SMF30PG CSECT
SMF30PG AMODE 31
SMF30PG RMODE ANY
MACRO
$STRGACQ &TYPE, &LEN=, &LOC=
AIF ( '&TYPE' EQ ' R' ).L0001
L R0, &LEN
AGO .L0002
.L0001 ANOP
.L0002 ANOP
A R0, =F' 4'
STORAGE OBTAIN, LENGTH=(Ø), LOC=&LOC
ST R0, Ø(R1)
LA R1, 4(R1)
MEND

MACRO
$STRGREL &TYPE, &ADDR=
AIF (' &TYPE' EQ 'R'). L0001
L R1, &ADDR
AGO .L0002

.L0001 ANOP
LR R1, &ADDR

.L0002 ANOP
S R1, =F'4'
L R0, Ø(R1)
STORAGE RELEASE, ADDR=(1), LENGTH=(Ø)
MEND

* YREGS
USING SMF30PG, R12

$CONS DS ØH
B $CONSL(R15)
DC C'SMF30PG &SYSDATE. &SYSTIME.'
DS ØH

$CONSL EQU ' - $CONS
STM R14, R12, 12(R13) Store caller information
LR R12, R15 R12 Base
LR R2, R1 Save R1 for later
$STRGACQ A, LEN=A($DYNL), LOC=31 Acq Dyn Stor
ST R13, 4(R1) Save Back pointer
ST R1, 8(R1) Save forward pointer
LR R13, R1 My save area
USING $DYN, R13 Let Assembler know
BAS R14, OPEN#ØØØ Open all the files.
LTR R15, R15 Mandatory files opened ?
BNZ RETURN4 N: bail out
LA R1, SYSIN point to SYSIN offset
M R0, =A(FILESIZE) Get the real offset
A R1, FILEDATA Add start
MVI INCL, INCLALL Default to include all
CLI FILEIND-FILEINFO(R1), X'Ø0' Was the file opened ?
BE SMFP#Ø2Ø N:
L R2, FILEREC-FILEINFO(), R1 Get the record address
L R3, FILEDCB-FILEINFO(), R1 Get the SYSIN DCB Addr

SMFP#Ø5 DS ØH
LR R0, R2 Move record ADDR to R0
GET (3) Get a record
CLC Ø(Ø8, R2), =C'JOBNAME=' Is it a mask ?
BNE SMFP#ØØØ N:
CLI Ø8(R2), =C' ' Is there a mask ?
BE SMFP#ØØØ N:
INDICATE MASK PRESENT

ACQ STORAGE FOR MASK

STORE NEXT ADDRESS

STORE MASK

STORE ANCHOR

STORE MASK

LENGTH OF JOBNAME

SUBTRACT START ADDRESS

IS IT ZERO?

STORE LENGTH

READ NEXT RECORD

PROCESS SMF FILE

IS THE FILE OPEN?

GET A RECORD

INDICATE THIS IS A STEP RECORD

IS THIS A SUBTYPE 4?

IS THIS A SUBTYPE 5?

INDICATE THIS IS A SECTION?

GET THE END DATE

CONVERT DATE

SAVE DATE FOR LATER

GET THE END TIME

CONVERT TIME

SAVE TIME FOR LATER

IDENTIFICATION SECTION PRESENT?
BE    SMFP#026
CLC   SMF30LN- SMFRCD30(2, R2),=H'0' Iden section present ?
BE    SMFP#026
CLC   SMF30ON- SMFRCD30(2, R2),=H'0' Iden section present ?
BE    SMFP#026
L     R4, SMF30OF- SMFRCD30(R2)  map the iden section
AR    R4, R2
MVC   JOBNAME, SMF30JBN- SMF30ID(R4)  get jobname
CLI   INCL, INCLALL  is this include all ?
BE    SMFP#254  n:
L     R5, MASKLIST  point to mask list
SMFP#253 DS    ØH
LTR   R5, R5  is one present ?
BZ    SMFP#254  n: get another record
BAS   R14, CHEK#000  check the mask
LTR   R15, R15  check good ?
BZ    SMFP#254  y: process record
L     R5, MASKNEXT-MASKDSCT(, R5) get next mask
B     SMFP#253  check again
SMFP#254 DS    ØH
MVC   J OBNUM, SMF30JNM- SMF30ID(R4)  get the job number
MVC   STEPNAME, SMF30STM- SMF30ID(R4)  get the step name
MVC   PROGRAM, SMF30PGM- SMF30ID(R4)  get the program name
LA    R1, SMF30STD- SMF30ID(R4)  Get start date
BAS   R14, CNVD#000  Convert date
MVC   STRDTDE, WORKDATE+2  Save for later
ICM   R1, 15, SMF30STI- SMF30ID(R4)  Get start time
BAS   R14, CNVT#000  Convert Time
MVC   STRTTME, WORKTIME+2  Save time for later
SMFP#026 DS    ØH
BAS   R14, HEDR#000  Build header info
BAS   R14, SUDT#000  Build SU details
BAS   R14, TMDT#000  Build Time details
BAS   R14, STDT#000  Build Storage details
B     SMFP#025  Get next record
SMFP#030 DS    ØH
SMFP#035 DS    ØH
SMFP#040 DS    ØH
B    RETURNØ
RETURNØ DS    ØH
LA    R5, 0  Process OK. RC=0
B    RETURN
RETURN4 DS    ØH
LA    R5, 4  Process not OK. RC=4
RETURN DS    ØH
BAS   R14, CLOS#000  Close all files.
LR    R1, R13  Save current dyn area
L     R13, 4(, R13)  Restore old dyn area
$STRGREL R, ADDR=1  Release curr dyn area
L     R14, 12(, R13)  Restore R14
LR R15, R5 R15 - > RC
LM R0, R12, 20(R13) Restore other regs
BSM 0, R14 goback
* Check if the pattern mask matches the jobname

CHEK#000 BAKR R14, 0
LA R2, MASKNAME - MASKDSCT(R5) Get address of Mask
LA R3, MASKNAML - MASKDSCT(R5) Point to length of mask
LA R4, JOBNAME Point to current jobname
ASAXWC PATTERNSTR=(2),
        PATTERNSTRL=(3),
        STRING=(4),
        STRINGLEN==A(8),
        ZEROORMORE=Z,
        ONECHAR=O,
        RETCODE=RETCODE,
        WORKAREA=WORKAREA,
        MF=(E, MYLIST)
L R15, RETCODE RC=0, match, else nomatch
CHEK#999 PR Go back to caller
*
OPEN#000 BAKR R14, 0
SLR R0, R0 Initialize R0
LA R1, FILESIZ Get the size of 1 file rec
LA R3, DDINFOS Get no. of DDs
MR R0, R3 Get size required
LR R0, R1
$STRGACQ R, LEN=Ø, LOC=31 Acq Dyn Stor
ST R1, FILEDATA Store returned addr
SLR R2, R2
LA R3, DDINFO Point to DDINFO
LR R4, R1
LA R5, DDINFOS Get no. of DDs
USING FILEINFO, R4
OPEN#005 DS ØH
LA R0, MODOPENL Get length of OPEN MODEL
$STRGACQ R, LEN=Ø, LOC=31 ACQ dyn area
ST R1, FILEOPEN Store returned addr
MVC Ø(MODOPENL,R1), MODOPEN Move OPEN MODEL
LA R0, MODCLOSL Get Length of CLOSE MODEL
$STRGACQ R, LEN=Ø, LOC=31 ACQ dyn area
ST R1, FILECLOS Store returned addr
MVC Ø(MODCLOSL,R1), MODCLOS Move close model
LA R0, MODDCBEL Get length of DCBE model
$STRGACQ R, LEN=Ø, LOC=31 ACQ dyn area
ST R1, FILEDCBE Store returned addr
MVC Ø(MODDCBEL,R1), MODDCBE Move DCBE model
USING DCBE, R1
MVC DCBEEODA, 16(R3) Save EOD address
MVC DCBESYNA, 20(R3) Save SYNAD address
DROP R1
CLI 8(R3), C'I'
BNE OPEN#010
y:
LA R0, MODDCBIL
$STRGACQ R, LEN=Ø, LOC=24
ST R1, FILEDCB
MVC Ø(MODDCBIL, R1), MODDCBIL
B OPEN#015
OPEN#010 DS ØH
LA R0, MODDCBOL
$STRGACQ R, LEN=Ø, LOC=24
ST R1, FILEDCB
MVC Ø(MODDCBOL, R1), MODDCBOL
B OPEN#015
USING IHADCB, R1
MVC DCBDDNAM, Ø(R3)
MVC DCBDCBE, FILEDCBE
DROP R1
L R0, 12(, R3)
$STRGACQ R, LEN=Ø, LOC=31
ST R1, FILEREC
MVC FILERECL, 12(R3)
XC FILEIND, FILEIND
MVC FILEHEDR, 24(R3)
L R6, FILEDCB
L R7, FILEOPEN
CLI 8(R3), C'I'
BNE OPEN#020
n:
OPEN ((6), INPUT), MF=(E, (7)), MODE=31 open file
B OPEN#025
OPEN#020 DS ØH
OPEN ((6), OUTPUT), MF=(E, (7)), MODE=31 open file
OPEN#025 DS ØH
LTR R15, R15
BZ OPEN#035
Y:
CLI 9(R3), C'M'
BNE OPEN#040
n:
LA R2, 4
B OPEN#040
OPEN#035 DS ØH
MVI FILEIND, X'01'
CLI 8(R3), C'I'
BE OPEN#040
y:
LR R1, R4
BAS R14, HDRW#ØØØ
OPEN#040 DS ØH
LA R3, DDINFOL(, R3)
LA R4, FILESZ(, R4)
BCT R5, OPEN#005
open them
LR R15, R2
OPEN#999 PR
DROP R4

* Write header records to output file
HDRW#000 BAKR R14, 0

LR R4, R1

Point to offset
L R0, FILEREC-FILEINFO(R4)
Get record addr
L R3, FILEDCB-FILEINFO(R4)
Get DCB addr
L R1, FILEDCCL-FILEINFO(R4)
Get record length
L R14, FILEHEDR-FILEINFO(R4)
Get header rec addr

LR R15, R1

length in R15
ICM R15, B'1000', BLANK
Default to ' '
LTR R14, R1
Is header present?
BZ HDRW#999
n: bail out
MVCL R0, R14
Move record to output
L R0, FILEREC-FILEINFO(R4)
point to record addr
PUT (3)
WRite output

HDRW#999 PR
goback

* Close all files.
CLOS#000 BAKR R14, 0

L R2, FILEDATA
Point to file information.
LA R3, DDIINFOS
Get number of DDs
USING FILEINFO, R2
CLOS#005 DS ØH

L R6, FILEDCB
Get DCB addr
L R7, FILECLOS
Get close area
CLI FILEIND, X'00'
Is this file open?
BE CLOS#010
n:
CLOSE ((6)), MF=(E,(7)), MODE=31
Close file

CLOS#010 DS ØH

L R1, FILEOPEN
Point to open area
$STRGREL R, ADDR=1
Release storage
L R1, FILECLOS
Point to close area
$STRGREL R, ADDR=1
Release storage
L R1, FILEDCB
Point to dcb area
$STRGREL R, ADDR=1
Release storage
L R1, FILEDCBE
Point to dcbe area
$STRGREL R, ADDR=1
Release storage
L R1, FILEREC
Point to rec area
$STRGREL R, ADDR=1
Release storage
MVI FILEIND, X'00'
Indicate file closed
LA R2, FILESIZ(R2)
Bump to next
BCT R3, CLOS#005
Close them

CLOS#999 PR
go back
DROP R2

PRTF#000 BAKR R14, 0

Write output to files
M R0 =A(FILESIZ)
R1 points to offset
A R1, FILEDATA
add start offset
CLI FILEIND-FILEINFO(R1), X'00'
is this file open?
BE PRTF#999  n: goback
L R2,FILEREC-FILEINFO(R1)  Point to record addr
L R3,FILEDCB-FILEINFO(R1)  Point to DCB
L R4,FILERECL-FILEINFO(R1) Get record length
LR R0, R2
LR R1, R4
LA R14, PRTLINE  Point to PRTLINE
LA R15, L'PRTLINE  Get PRTLINE address
ICM R15, B'1000', BLANK Default to ' '
MVCL R0, R14 Move record to output
LR R0, R2 R0-> record
PUT (3) Put to record to file/.
PRTF#999 PR goback
HEDR#000 BAKR R14, Ø write header part of record
MVC PRTLINE(L'BLANK), BLANK initialize prtline
MVC PRTLINE+1(L'PRTLINE-1), PRTLINE
LA R1, PRTLINE point to prtline
USING HEDR, R1 let assembler know
MVC HEDRJNME, JOBNAME move jobname
MVC HEDRJNUM, JOBNUM move jobnumber
MVC HEDRSTDT, STRTDTE move start date
MVC HEDRSTTM, STRTTME move start time
MVC HEDRENDT, ENDDATE move end date
MVC HEDRENTM, ENDTIME move end time
CLI STEP, ISJOB is this step info ?
BE HEDR#999 n: goback
MVC HEDRSTNM, STEPNAME move stepname
MVC HEDRPGNM, PROGRAM move program name
HEDR#999 PR goback
SUDT#000 BAKR R14, Ø write SU details
BAS R14, HEDR#000 write header part of rec
USING SMFRCDO30, R2 let assembler know
CLC SMF30POF, =F‘Ø’ is there info ?
BE SUDT#999 n: goback
CLC SMF30PLN, =H‘Ø’ is there info ?
BE SUDT#999 n: goback
CLC SMF30PON, =H‘Ø’ is there info ?
BE SUDT#999 n: goback
A R2, SMF30POF Add offset to section.
USING SMF30PRF, R2 Let assembler know
LA R1, PRTLINE Point to prtline
CLI STEP, ISJOB is this jobinformation ?
BE SUDT#005 y:
LA R1, HEDRSTEP(, R1) Bump step header info
LA R3, STEPSU indicate stepsu
B SUDT#010
SUDT#005 DS ØH
LA R1, HEDRJOB(, R1) Bump past job header info
LA R3, JOBSU indicate jobsu
SUDT#Ø10 DS ØH
USING SUDSECT, R1                Let assembler know
MVC SUTOT, EDMASKN               Move mask to su totals
MVC SUCPU, EDMASKN               Move mask to su cpu
MVC SUSRB, EDMASKN               Move mask to su srb
MVC SUIO, EDMASKN                Move mask to su io
ICM R14, 15, SMF3ØSRV           get total su
CVD R14, DWORD                  convert to decimal
ED SUTOT, DWORD+3               edit to su totals
ICM R14, 15, SMF3ØCSU           get cpu su
CVD R14, DWORD                  convert to decimal
ED SUCPU, DWORD+3               edit to su cpu
ICM R14, 15, SMF3ØSRB           get srb su
CVD R14, DWORD                  convert to decimal
ED SUSRB, DWORD+3               edit to su srb
ICM R14, 15, SMF3ØIO            get io su
CVD R14, DWORD                  convert to decimal
ED SUIO, DWORD+3               edit to su io
CLI STEP, ISSTEP               is this step info ?
BE SUDT#Ø15              y: goback
MVC SULWM, SMF3ØWLM            move wlm class
MVC SUSRVM, SMF3ØSCN          move wlm service name
MVC J OBLMC, SMF3ØWLM          move wlm class
MVC J OBLM, SMF3ØSCN          move wlm service name
SUDT#Ø15 DS ØH
LR R1, R3                     point to dd offset
BAS R14, PRTF#Ø000          write this record
SUDT#999 PR                     goback
DROP R1, R2
TMDT#Ø00 BAKR R14, 0         write TIME details
BAS R14, HEDR#Ø000       write header part of rec
USING SMFRCRD3Ø, R2       let assembler know
CLC SMF3ØCOF, =F'Ø'        is there info ?
BE TMDT#999             n: goback
CLC SMF3ØCLN, =H'Ø'        is there info ?
BE TMDT#999             n: goback
CLC SMF3ØCON, =H'Ø'        is there info ?
BE TMDT#999             n: goback
ZAP DEXCP, =P'Ø'            Initialize excp count
CLC SMF3ØUOF, =F'Ø'        is there info ?
BE TMDT#Ø03           n: goback
CLC SMF3ØULN, =H'Ø'        is there info ?
BE TMDT#Ø03           n: goback
CLC SMF3ØUON, =H'Ø'        is there info ?
BE TMDT#Ø03           n: goback
LR R3, R2
A R3, SMF3ØUOF
L R1, SMF3ØTEP-SMF3ØURA(R3) Get excp count
CVD R1, DEXCP                Convert to decimal
TMDT#003 DS ØH
   A   R2,SMF30COF
   USING SMF30CAS,R2
   LA   R4,PRTLINE
   CLI  STEP,ISJOB
   BE   TMDT#005
       LA   R4,HEDRSTEP(,R4)
       LA   R3,STEPTIME
TMDT#005 DS ØH
   LA   R4,HEDRJOB(,R4)
   LA   R3,JOBTIME
TMDT#010 DS ØH
   USING TIMEDSCT,R4
   MVC TIMEEXCP,EDMASKN
   ICM R1,15,SMF30CPT
   BAS R14,CNVT#000
   MVC TIMECPU,WORKTIME+2
   ICM R1,15,SMF30CPS
   BAS R14,CNVT#000
   MVC TIMESRB,WORKTIME+2
   ED TIMEEXCP,DEXCP+3
   CLI STEP,ISSTEP
   BE TMDT#015
   MVC TIMEWLM,JOBWLMC
   MVC TIMESRVN,JOBWLMS
TMDT#015 DS ØH
   LR   R1,R3
   BAS R14,PRTF#000
TMDT#999 PR
   DROP R4,R2
STDT#000 BAKR R14,Ø
   BAS R14,HEDR#000
   USING SMFRCD30,R2
   CLC SMF30ROF,=F'Ø'
   BE STDT#999
       n: goback
   CLC SMF30RLN,=H'Ø'
   BE STDT#999
       n: goback
   CLC SMF30RON,=H'Ø'
   BE STDT#999
       n: goback
   A   R2,SMF30ROF
   USING SMF30SAP,R2
   LA   R4,PRTLINE
   CLI STEP,ISJOB
   BE STDT#999
       y: goback
   LA   R4,HEDRSTEP(,R4)
   LA   R3,STEPSTOR
   USING STORDSCT,R4
   MVC STORUB16,EDMASKN

Move edit mask to UStor<16M
MVC STORUA16, EDMASKN
Move edit mask to USStor>16M
MVC STORSB16, EDMASKN
Move edit mask to SSStor<16M
MVC STORSA16, EDMASKN
Move edit mask to SSStor>16M
ICM R14, B'1111', SMF3ØURB
Get Ustor<16M
SRL R14, 10
Convert to KB
CVD R14, DWORD
Convert to decimal
ED STORUB16, DWORD+3
Edit to output area
ICM R14, B'1111', SMF3ØEUR
Get Ustor>16M
SRL R14, 10
Convert to KB
CVD R14, DWORD
Convert to decimal
ED STORSA16, DWORD+3
Edit to output area
ICM R14, B'1111', SMF3ØARB
Get Sstor<16M
SRL R14, 10
Convert to KB
CVD R14, DWORD
Convert to decimal
ED STORSB16, DWORD+3
Edit to output area
ICM R14, B'1111', SMF3ØEAR
Get Sstor>16M
LR R1, R3
Point to dd offset
BAS R14, PRTF#000
Write this output rec
STDT#999 PR
Goback
DROP R4, R2
CNVT#000 BAKR R14, 0
Convert time to HH:MM:SS
SLR R0, R0
R1-> time in 100th of sec
D R0, =F'360000'
Get hours
CVD R1, DWORD
Convert to decimal
SRP DWORD, 4, 0
0000000000HH0000C
ZAP WTME, DWORD
Save it for later
LR R1, R0
Get reminder
SLR R0, R0
D R0, =F'6000'
Get minutes
CVD R1, DWORD
Convert to decimal
SRP DWORD, 2, 0
000000000000MM00C
AP WTME, DWORD
Add to saved time
LR R1, R0
Get reminder
SLR R0, R0
D R0, =F'100'
Get seconds
CVD R1, DWORD
Convert to decimal
AP WTME, DWORD
Add to saved time
MVC WORKTIME, EDMASKT
Move edit mask for time
ED WORKTIME, WTME+4
Edit time
CNVT#999 PR
Goback
* Convert date from ØnYYDDDF to CCYY-MM-DD format
CNVD#000 BAKR R14, 0
ZAP DWORD, 0(4, R1)
R1 points to input date
OI DWORD+L'DWORD-1, X '0F'
Make pack!
AP DWORD, =P'1900000'
Add 1900000
ZAP WDATE, DWORD
Save for later
SRP  DWORD, 64-3, Ø  Remove days
CVB  R1, DWORD  Convert to binary
SLR  RØ, RØ
D   RØ, =F'4'
C   RØ, =F'0'
BE  CNVD#005  y: is leap year
L   R1, =A(LEAPYR)  not a leap year
B   CNVD#010
CNVD#005 DS  ØH
L   R1, =A(LEAPYR)  is a leap year
CNVD#010 DS  ØH
ZAP  DWORD, WDATE  restore date
SRP  WDATE, 64-3, Ø  remove days
SRP  WDATE, 4, 0  00000000CCYY0000F
SRP  DWORD, 12, 0  DDD00000000000DF
SRP  DWORD, 64-12, Ø  0000000000000DDDF
AP  WDATE, =P'100'  Indicate 1st month
CNVD#015 DS  ØH
CP  DWORD, Ø(2, R1)  is the days < table
BNH  CNVD#020  n:
AP  WDATE, =P'100'  Add 1 month
SP  DWORD, Ø(2, R1)  Subtract date in month
LA  R1, 2(, R1)  Bump to next table entry
B   CNVD#015  loop back
CNVD#020 DS  ØH
AP  WDATE, DWORD  Add date
MVC  WORKDATE, EDMASKD  Move mask for date
ED  WORKDATE, WDATE+3  Edit date
CNVD#999 PR  goback
*
\begin{verbatim}
Literals
\end{verbatim}
Z DC  CL1 ' '  zero or more char
O DC  CL1 '?'  exactly 1 char
LEAPYR DC  P'31,29,31,30,31,30,31,30,31,30,31,30,31,30,31'  leap table
NLEAPYR DC  P'31,28,31,30,31,30,31,30,31,30,31,30,31,30,31'  leap table
EDMASKN DC  X'4021202022A220222A212020'  Edit mask for number
EDMASKT DC  X'4021202022A220222A212020'  Edit mask for time
EDMASKD DC  X'4021202022A220222A212020'  Edit mask for date
BLANK DC  C' '  Blanks
DS  ØF  DD table
DDINFO DC  C'SYSIN ', C'I', C'O', A(8Ø), A(SMFP#010), A(SMFP#015)
DC  A(0)
SYSIN EQU  Ø
DDINFOŁ EQU  '*' DDINFO
DC  C'SMFN ', C'I', C'M', A(60000), A(SMFP#030), A(SMFP#035)
DC  A(Ø)
SMFIN EQU  1
DC  C'STEPSU ', C'O', C'O', A(133), A(Ø), A(Ø)
DC  A(SUSHDR)
STEPSU EQU  2
\end{verbatim}

DC C' JOBSU ', C' O', C' O', A(133), A(0), A(0)
DC A(SUJ HDR)
JOBSU EQU 3
DC C' STEPTIME', C' O', C' O', A(133), A(0), A(0)
DC A(TM HDR)
STEPTIME EQU 4
DC C' JOBTIME ', C' O', C' O', A(133), A(0), A(0)
DC A(STSHDR)
JOBTIME EQU 5
DC C'STEPSTOR', C' O', C' O', A(133), A(0), A(0)
DC A(STSHDR)
STEPSTOR EQU 6
DDINFOS EQU (*-DDINFO)/DDINFOL
DC X'FFFFFFFF'
MODDCBO DCB DCBE=MODDCBE, MACRF=(PM), DSORG=PS, LRECL=133 output DCB
MODDCBOL EQU *-MODDCBO
MODDCBI DCB DCBE=MODDCBE, DDNAME=Ø, MACRF=(GM), DSORG=PS Input DCB
MODDCBIL EQU *-MODDCBI
MODDCBE DCBE RMODE31=BUFF, EODAD=Ø, SYNAD=Ø Model DCBE
MODDCBEL EQU *-MODDCBE
MODOPEN OPEN ( ), MF=L, MODE=31 Model OPEN
MODOPENL EQU *-MODOPEN
MODCLOS CLOSE ( ), MF=L, MODE=31 Model CLOSE
MODCLOS EQU *-MODCLOS
*
LTORG
SMF3ØPG CSECT
SUSHDR DS ØF
DC C' JobName JobNum Start Date St Time End D'
DC C'ate End Time StepName Program Total SU'
DC C' CPU SU SRB SU IO SU'
SUJ HDR DS ØF
DC C' JobName JobNum Start Date St Time End D'
DC C'ate End Time StepName Program Total SU CPU SU SRB'
DC C' SU IO SU WLM Cls Serv Name'
TMHDR DS ØF
DC C' JobName JobNum Start Date St Time End D'
DC C'ate End Time StepName Program TCB CPU SRB'
DC C' B CPU EXCP'
TMJ HDR DS ØF
DC C' JobName JobNum Start Date St Time End D'
DC C'ate End Time TCB CPU SRB CPU EXCP'
DC C' WLM Cls Serv Name'
STSHDR DS ØF
DC C' JobName JobNum Start Date St Time End D'
DC C'ate End Time StepName Program USR<16(K)'
DC C' USR>16(K) SYS<16(K) SYS>16(K)'
$DYN DSECT Dynamic area
SAVEAREA DS 18F Save area
FILEDATA DS F       Pointer to file information
MASKLIST DS F       Pointer to mask list
RETCODE DS F        Return code
DWORD DS D           work
DEXCP DS D           work excp count
WDATE DS D           work date
TIME DS D            work time
WORKTIME DS CL(L'EDMASKT) work area
WORKDATE DS CL(L'EDMASKD) work area
JOBWLMC DS CL08     wlm class
JOBWLM5 DS CL08     wlm service name
STRTDTE DS CL10     Start date
ENDDATE DS CL10     End date
STRTIME DS CL08     Start Time
ENDTIME DS CL08     End time
JOBNAME DS CL08     Jobname
JOBNUM DS CL08      Job Number
STEPNAME DS CL08    Step Name
PROGRAM DS CL08     Program name
STEP DS X           Step indicator
ISSTEP EQU X'00'    ...Is step information
ISJOB EQU X'01'     ...Is job information
INCL DS X           Include indicator
INCLALL EQU X'01'   ...Include all
PRTLINE DS CL133    Print line

FILEINFO DSECT      DSECT for file area
FILEOPEN DS F       Open area addr
FILECLOS DS F       Close area addr
FILEDCB DS F        DCB addr
FILEDCBE DS F       DCBE addr
FILEREC DS F        rec addr
FILERECL DS F       rec length
FILEHEDR DS F       header addr
FILEIND DS X        indicator
DS CL3              filler
FILESZ EQU *-FILEINFO length of file area
HEDR DSECT          Header information for rec
DS CL1
HEDRJNME DS CL8     Jobname
DS CL1
HEDRJNUM DS CL8     Jobnumber
DS CL1
HEDRSTDT DS CL10    Start date
DS CL1
HEDRSTTM DS CL08    Start Time
DS CL1

HEDRENDT DS CL10  End date
   DS CL1
HEDRENTM DS CL08  End time
   DS CL1
HEDRJOB EQU '*-HEDR  Length of job header
HEDRSTNM DS CL08  Step Name
   DS CL1
HEDRPGNM DS CL08  Program Name
   DS CL1
HEDRSTEP EQU '*-HEDR  Length of Step Header
HEDRL EQU '*-HEDR  Total length of header
SUDSECT DSECT  Dsect for SU details
SUTOT DS CL10  Total SU units
   DS CL1
SUCPU DS CL10  CPU SU units
   DS CL1
SUSRB DS CL10  SRB SU units
   DS CL1
SUIO DS CL10  IO su units
   DS CL1
SUWLM DS CL08  WLM class name
   DS CL1
SUSRVNM DS CL08  WLM serv class
SUL EQU '*-SUDSECT  Length
TIMEDSCT DSECT  Dsect for TIME details
TIMECPU DS CL08  TCB CPU time
   DS CL1
TIMESRB DS CL08  TCP SRB time
   DS CL1
TIMEEXCP DS CL10  Total EXCP
   DS CL1
TIMEWLM DS CL08  WLM class name
   DS CL1
TIMESRVN DS CL08  WLM serv class
TIMEL EQU '*-TIMEDSCT  Length
STORDSCT DSECT  Dsect for Storage section
STORUB16 DS CL10  User stor < 16M
   DS CL1
STORUA16 DS CL10  User stor > 16M
   DS CL1
STORSB16 DS CL10  Sys stor < 16M
   DS CL1
STORSA16 DS CL10  Sys Stor > 16M
   DS CL1
STORL EQU '*-STORDSCT  Length
MASKDSCT DSECT  Dsect for Mask details
MASKNEXT DS F  Ptr to Next mask info elem
MASKNAML DS F  Length of Mask
MASKNAME DS CL8  Name of Mask
MASKL EQU '*-MASKDSCT  Length
DCBD DSORG=PS, DEVD=DA
Analysing data-in-virtual statistics

INTRODUCTION

Since its introduction a long time ago (with MVS/XA!), when it received some attention, DIV (Data-In-Virtual) seems to have fallen into oblivion. The main reason for that is the fact that DIV is somewhat difficult to use because the Assembly-language primitive functions one must use are not readily available in high-level languages. However, DIV, which is a set of primitive functions, enables an application program to load and manage substantial amounts of data into memory from a VSAM Linear DataSet (LDS). The LDS itself can grow to 4GB and the program can map up to (almost) 2GB of it at a time in central memory. Applications can create, read, and update data without the I/O buffer, blocksize, and record considerations that the traditional GET and PUT types of access method require. An application written for data-in-virtual views its permanent storage data as a seamless body of data without internal record boundaries. Among the applications that can be considered for a data-in-virtual implementation are applications that process large arrays, VSAM relative record applications, and BDAM fixed-length record applications. The potential benefits may be realized eventually as DIV merges with hiperspaces (a related concept) and as subsystems, languages, and application packages exploit the DIV benefits. For example, DIV is used by DFSMS when I/O to SMS control datasets is needed.
The data-in-virtual services process the application data in 4096-byte (4KB) units on 4KB boundaries called blocks. The application data resides in what is called a data-in-virtual object, a data object, or simply an object. The data-in-virtual object is a continuous string of uninterrupted data. When one writes an application using the techniques of data-in-virtual, the I/O takes place only for the data referenced and saved. Only the referenced pages of such an object are brought into virtual storage. Bytes of the mapped pages can be accessed and changed in normal program execution without regard to the need for updating. On request, or at the time the connection to the DIV object is terminated, only the changed pages are written back to the linear dataset. If one runs an application using conventional access methods, and then runs it again using data-in-virtual, one will notice a difference in performance. This difference depends on both the size of the dataset and its access pattern. To gain the right to view or update the object, an application must use the ACCESS service. ACCESS is similar to the OPEN macro of VSAM. It has a mode parameter of READ or UPDATE, and it gives your application the right to read or update the object. If the application has finished processing the object, it uses UNACCESS to relinquish access to the object.

Before using the DIV macro to process a linear dataset object (or a hiperspace object), one must create the dataset (or the hiperspace). OS/390 MVS Programming: Authorized Assembler Services Guide (GC28-1763) explains how to use DIV macro functions. The ‘how to’ reference for hiperspaces is the Extended Addressability Guide, (GC28-1468). Also, it is worth consulting An Introduction to Data-in-Virtual (GG66-0259), which may be a bit old but provides a few Assembler, Fortran, and PL/I examples.

COLLECTING DIV DATA
When enabled by the SMFPRMxx TYPE parameter, SMF creates record type 41, which provides resource usage information regarding data-in-virtual. There are two subtypes of this SMF record: subtype 1 is an ACCESS record – the ACCESS data
section is written when a DIV object is accessed; subtype 2 is an UNACCESS record – the counts for the I/O activity section are accumulated by data-in-virtual while the object is in use and are reported at the time of the UNACCESS request. The subtype 2 record is written whenever a data-in-virtual object is unaccessed.

A detailed description of layout of SMF type 41 record and its subtypes can be obtained from the MVS System Management Facilities (SMF) (SA22-7630-03) manual. You can also find the subtype descriptions in the macro ITVSMF41 in SYS1.MACLIB.

Based on record descriptions obtained from above mentioned manual a simple DIV report writer was written.

**CODE**

The code is a two part stream. In the first part (COPY412), selected SMF records (selection being defined by INCLUDEs condition) are copied from an SMF dataset to a VB file, which is the input file for the second step.

In the second step (DIV412), the captured records are formatted by invoking REXX EXEC (DIV412), and a report produced. The report shows the users performing access/unaccess of an object, along with the timestamp of when the object was accessed/unaccessed, the size of the object, and its read/write/re-read count data. Elapsed time during which the object was read or updated is calculated from two TOD timestamps.

**JCL**

```plaintext
//DIVJOB    JOB ACCT#, 
//          MSGLEVEL=(1,1), 
//          MSGCLASS=R, 
//          NOTIFY=&SYSUID
//COPY412  EXEC PGM=ICETOOL
//TOOLMSG  DD SYSOUT=* 
//DFSMSG   DD SYSOUT=* 
//SMF412   DD DSN=your.copied.by.sort.to.VB.smf.dataset, 
//         SPACE=(CYL,(1)),UNIT=SYSDA,DISP=(NEW,PASS), 
//         DCB=(RECFM=VB,LRECL=32756,BLKSIZE=32760)
```

DIV412 EXEC
/* REXX EXEC to read and format SMF record 41.2*/
ADDRESS TSO
/*-----------------------------------------------*/
/* Print report header and labels */
/*-----------------------------------------------*/
Out.1 = left(' ',30,' ')
     || center('Data in Virtual Report ',22,)
     || left(' ',15,' ')
Out.2 = left(' ',20,' ')
     || center('Report produced on',18,)
     || left(' ',1,' ')|| left(date(),11,)
     || left(' ',1,' ')|| left('at ',3,' ')
     || left(time(),10)
Out.3 = '  '  
Out.4 = left('SMF date',11) left('SMF time',9),
     | left('Sid',4) left('DFP lvl.',7),
     | left('Job name',9) left('DD name',9),
     | left('Access Time',15) left('Unaccess Time',15),
     | left('A.Size',6) left('U.Size',7),
     | left('Mode',5) left('Elaps.sec.',10),
     | left('Block',5) left('Block',6),
     | left('Read',5),
     | left('Write',5)
Out.5 = left(' ',118) left('read',5),
     | left('write',6) left('rread',6),
     | left('I/O',5) left('I/O',3)
Out.6 = LEFT('-',149,'-')
"EXECIO * DISKW DIV41 (STEM Out."
' EXECIO * DISKR SMF ( STEM x. FINIS"
do i = 1 to x.Ø
/* Header/Self-defining Section */
/* */
smftype  = c2d(SUBSTR(x.i,2,1))  /* SMF record type */
smfstype = c2d(SUBSTR(x.i,19,2)) /* Record subtype */
IF smftype = '41' Then
Do
  smfdate = SUBSTR(c2x(SUBSTR(x.i,7,4)),3,5) /* Unpack SMF date */
  smftime = smf(c2d(SUBSTR(x.i,3,4))) /* Decode SMF time */
  sysid   = SUBSTR(x.i,11,4)           /* System identification */
  trp     = c2d(SUBSTR(x.i,21,2))   /* number of triplets*/
  opd     = c2d(SUBSTR(x.i,25,4))   /* offset to product section*/
  lpd     = c2d(SUBSTR(x.i,29,2))   /* length of product section*/
  npd     = c2d(SUBSTR(x.i,31,2))   /* number of product sections*/
  od1     = c2d(SUBSTR(x.i,33,4))   /* offset to access section*/
  ld1     = c2d(SUBSTR(x.i,37,2))   /* length of access section*/
  nd1     = c2d(SUBSTR(x.i,39,2))   /* number of access sections*/
  od2     = c2d(SUBSTR(x.i,41,4))   /* offset to unaccess section*/
  ld2     = c2d(SUBSTR(x.i,45,2))   /* length of unaccess section*/
  nd2     = c2d(SUBSTR(x.i,47,2))   /* number of unaccess sections*/
  od3     = c2d(SUBSTR(x.i,49,4))   /* offset to i/o activity*/
  ld3     = c2d(SUBSTR(x.i,53,2))   /* length of i/o activity*/
  nd3     = c2d(SUBSTR(x.i,55,2))   /* number of i/o activity*/
/* Product Section */
/* */
IF opd <> Ø AND npd <> Ø Then do
  opd=opd-3
  dfplvl = SUBSTR(x.i,opd,8)           /* product level */
  prod   = SUBSTR(x.i,opd+8,16)         /* component name*/
end
/* Object ACCESS Data Section */
/* */
IF od1 <> Ø and nd1 <> Ø Then do
  od1=od1-3
  dda   = SUBSTR(x.i,od1,8)            /* object ddname*/
  aza   = c2d(SUBSTR(x.i,od1+8,4))    /* object size */
  ata   = SUBSTR(ct(c2x(SUBSTR(x.i,od1+12,4))),11,15) /* TOD */
  tya   = c2d(SUBSTR(x.i,od1+16,1))   /* object type */
  ama   = c2d(SUBSTR(x.i,od1+17,1))   /* access mode */
  jbn   = SUBSTR(x.i,od1+18,8)        /* jobname/started task */
  SELECT
    when ama=1 then mode='Read'
    when ama=2 then mode='Update'
  END
end
/* Object UNACCESS Data Section */
/* */
IF od2 <> Ø and nd2 <> Ø then do
    od2=od2-3
    
    zu = c2d(SUBSTR(x.i,od2,4))               /* object size*/
    utu = SUBSTR(ct(c2x(SUBSTR(x.i,od2+4,4))),11,15) /* TOD*/
end
/* Object I/O Activity Section */
/* Object Activity Section */
IF od3 <> Ø and nd3 <> Ø then do
    od3=od3-3
    
    brd = c2d(SUBSTR(x.i,od3,4))       /* tot.no. of reads*/
    bwr = c2d(SUBSTR(x.i,od3+4,4))     /* tot.no. of writes*/
    brr = c2d(SUBSTR(x.i,od3+8,4))     /* tot.no. of re-reads*/
    inc = c2d(SUBSTR(x.i,od3+12,4))    /* tot.no.of i/o for read*/
    ouc = c2d(SUBSTR(x.i,od3+16,4))    /* tot.no.of i/o for write*/
end
    timedif=dif(utu,ata)      /* how long the object was accessed*/
/* formatting and printing a DIV entry */
divout = left(Date('N',smfdate,'J'),11) left(smftime,9),
        left(sysid,4)  left(dfplvl,8)  left(jbn,8),
        left(dda,8)    right(ata,15)   left(utu,17),
        left(aza,6)    left(uzu,6)     left(mode,7),
        right(timedif,8),
        right(brd,4)   right(bwr,5)    right(brr,5),
        right(inc,5)   right(ouc,5)
    
PUSH divout
    "EXECIO 1 DISKW DIV41"
end
exit

SMF PROCEDURE
/* REXX - convert a SMF time */
arg time
time1 = time % 100
hh = time1 % 3600
hh = RIGHT("0"||hh,2)
mm = (time1 % 60) - (hh * 60)
mm = RIGHT("0"||mm,2)
ss = time1 - (hh * 3600) - (mm * 60)
ss = RIGHT("0"||ss,2)
otime = hh||"":"||mm||"":"||ss /* Compose SMF time*/
return notime

© 2003. Xerphon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.
CT PROCEDURE

/*
 * TOD timestamp is a 16-byte EBCDIC representat *
 * The BLSUXTOD proc is described in "z/OS *
 * V1R3 MVS IPCS Customization"
 */
/*
arg todtime
If todtime <> 'ØØØØØØØØØØØØØØØØ' Then
   Do
      TOD_Value = X2C(todtime)
      Returned_Date = '--------------------------'
      address LINKPGM "BLSUXTOD TOD_Value Returned_Date"
   End
Else
   Returned_Date = ''
Return Returned_Date

DIF PROCEDURE

/*
 * Dif: REXX subroutine to find the
 * difference between two timestamps, in this
 * case in seconds
 */
/*
arg time2,time1
parse var time2 h2 ':' m2 ':' s2 ':' t2
parse var time1 h1 ':' m1 ':' s1 ':' t1
tot2 = h2*3600 + m2*60 + s2
tot1 = h1*3600 + m1*60 + s1
es = tot2 - tot1
if es < 0 then es = es + 86400
eh = es % 3600
es = es / 3600
ex = es
em = es % 60
es = es / 60
/* et = right(eh, 2, 0)':'right(em, 2, 0)':'right(es, 2, 0) */
return ex

Mile Pekic
Systems Programmer (Serbia and Montenegro) © Xephon 2003
Waiting for datasets

We have a group of FTP datasets ‘pushed’ to us – the first part of the dataset name is known, but the trailing qualifier is variable. The following program is started by our automation package, and is the name of a volume to look on and some further variable (time-based) information that is then used for a ‘cvaffilt’ call, which passes back the names (if any) of datasets found that match our pattern. We then extract the dataset names and write them to a dataset that is passed to the next step (obviously this could be replaced by a WTO/WTOS, a command being issued, or any other relevant action). The program is written to look for three datasets, and will wait for five minutes for them to arrive.

WAIT4FTP PROGRAM

**WAIT4FTP - WAIT FOR INCOMING FTP DATASETS**

* WAIT4FTP: SCAN A VOLUME, USING A DSN PATTERN, TO SEE HOW MANY DATASETS ARE PRESENT. WAIT FOR UP TO 5 MINUTES TO SEE IF THREE HAVE ARRIVED. IF NOT, EXIT WITH RC.

* FOR FTP TRANSFER OF DATASETS.

* PARMS: TWO PARMs, BOTH MANDATORY, SEPARATED BY A COMMA.

* 1/ A VOLID (TO LOOK ON FOR THE DATASETS)

* 2/ 4 DIGITS, TO BE INSERTED INTO THE DATASET MASK. THIS MASK WILL MAKE UP A DATASET NAME, IN THE FORM:

* FTPXFER.DATA.FILE.PFTP.DNNNTY.*

* WHERE 'NNNN' ARE THE 4 DIGITS PASSED TO US.

* OUTPUT: DATASET CONTAINING THE NAMES OF THE DATASETS LOCATED, WRITTEN TO DDNAME 'DSNAMES'.

* * *

**PRINT NOGEN**

* HOUSEKEEPING...

**WAIT4FTP CSECT**
WAIT4FTP AMODE 31
WAIT4FTP RMODE 24
BAKR R14,Ø
LR R12,R15
LA R11,2048(R12)
LA R11,2048(R11)
USING WAIT4FTP,R12,R11
L R9,Ø(R1)
LTR R9,R9
BZ BADPARMS
CLC Ø(2,R9),=H’11’
BNE BADPARMS
CLI 8(R9),C’,’
BNE BADPARMS
MVC VOLID(6),2(R9)
MVC DSNSER(4),9(R9)
MVC WTO1+23(6),VOLID
MVC WTO1+35(L’DSNAME),DSNAME
WTO1 WTO ‘WAIT4FTP- SCAN XXXXXX FOR: .........................X
............’,ROUTCDE=11
**********************************************************************
* INITIALIZE BUFFER LIST HEADER (BFLH) AND ELEMENTS (BFLE)... *
**********************************************************************
INITBFL DS ØH
* LA R2,BFLHDEF
LA R3,BFLSIZE
XR R5,R5
MVCL R2,R4
* LA R1,BFLHDEF
USING BFLMAP,R1
MVI BFLHNOE,BUFFNUM
OI BFLHFL,BFLHDSCB
LA R2,BFLHDEF+BFLHLN
USING BFLE,R2
LA R3,DSCBDEF
LA R4,BUFFNUM
BFLEINIT DS ØH
* OI BFLEFL,BFLECHR
MVI BFLELTH,DSCBSIZE
ST R3,BFLEBUF
LA R2,BFLELN(R2)
LA R3,DSCBSIZE(R3)
BCT R4,BFLEINIT
DROP R1,R2
**********************************************************************
* INITIALIZE THE FILTER CRITERIA LIST (FCL) HEADER AND ELEMENT... *
**********************************************************************
XC FCLDEF(FCLSIZE),FCLDEF
LA   R1, FCLDEF               R1 -> FCL HEADER
USING FCLMAP, R1              TEMP ADDRESSABILITY TO FCL
MVC FCLI D, =C' FCL '           SET EYECATCHER 'FCL '
MVC FCLCOUNT, =H'1'            SET NUMBER OF FCL ELEMENTS = 1
OI FCL1FLAG, FCL1EQF1          RETURN ONLY FORMAT1 DSCBS
LA   R2, FCHDEND               R2 -> 1ST (ONLY) FCL ELEMENT
USING FCLDSN, R2              TEMP ADDRESSABILITY
MVC FCLDSNLG, DSNAMELN        SET LENGTH OF DSN PATTERN
LA   R3, DSNAME                GET DSN PATTERN ADDRESS
ST   R3, FCLDSNA              ...SAVE IN FCL
DROP R1, R2
***********************************************************************
* SCAN THROUGH THE UCBS, LOOKING FOR OUR VOLUME.                      *
***********************************************************************
SCANVOLS DS ØH
USING UCBBOB, R4               ADDRESSABILITY TO UCB
LA   R4, UCBAREA               LOCATE UCB WORKAREA
XC   UCBWORK, UCBWORK         +INITIALIZE UCBSCAN WORKAREA
UCBLOOP DS ØH
  UCBSCAN COPY,                      X
    WORKAREA= UCBWORK,            X
    UCBAREA= UCBAREA,            X
    DCEAREA= NONE,               X
    DCELEN= Ø,                  X
    VOLSER= VOLID,              SELECT BY VOLSER X
    DEVN= Ø,                   START WITH FIRST UCB X
    DYNAM= YES,                 INCLUDE DYNAMICALLY ADDED UCBS X
    RANGE= ALL,                *ALL* UCBS (3&4 DIGIT) X
    NONBASE= NO,               X
    DEVCLASS= DASD,             DASD ONLY X
    DEVCID= Ø,                  DON'T SELECT BY DEVICE CHAR. X
    IOCTOKEN= NONE,            NO IODEVICE TABLE TOKEN X
    LINKAGE= SYSTEM,            USE PC CALL X
    PLISTVER= MAX
  *
LTR   R15, R15                  GOT UCB OK?
BZ    GETREAL                  YES..GET THE REAL UCB ADDRESS
  C    R15, =F'4'               END OF UCBS?
BE    NOTFND                   YES..SAY WE COULDN'T FIND VOL
  B    BADCALL                  NO...ERROR
***********************************************************************
* NOW WE HAVE OUR VOLUME - GET THE *REAL* UCB ADDRESS BY CALLING THE  *
* "UCBLOOK" MACRO...                                                   *
***********************************************************************
GETREAL DS ØH
  *
    MODESET KEY= ZERO, MODE= SUP            SUPERVISOR STATE FOR "UCBLOOK"
    *
    UCBLOOK UCBLOOK VOLSER= UCBVOLI,     USE VOLID FROM UCB COPY    X
UCBPTR=UCBADDR,       SAVE UCB ADDRESS IN HERE       X
LOC=ANY,                 UCB CAN BE ANYWHERE       X
NOPIN,                   DON'T PIN UCB                  X
RANGE=ALL,               3&4 DIGIT UCBS                 X
DEVCCLASS=DASD,          MAKE SURE ITS A DISK!         X
DYNAMI C=YES             CHECK DYNAMIC UCBS

*      ST  R15,SAVER15         SAVE RC
*      MODESET KEY=NZERO,MODE=PROB    BACK TO NORMAL
*      L     R15,SAVER15        RELOAD RC
LTR   R15,R15               OK?
BNZ   LOOKERR                NO...PANIC...
***********************************************************************
* NOW WE HAVE THE VOLUME WE WANT - ISSUE A CVAFFILT 'ACCESS=READ'     *
* REQUEST TO SEE IF THERE ARE ANY RELEVANT FTP DATASETS, AND, IF SO,  *
* HOW MANY...                                                       *
***********************************************************************
   LA    R7,5                WAIT UP TO 5 TIMES
   CVPLLOOP DS ØH
*      CVPL   CVAFFILT ACCESS=READ,                                      X
               FCL=FCLDEF,                                              X
               BUFLIST=BFLHDEF,                                         X
               UCB=UCBADDR
*      C     R15,F4              WAS RC=4?
BE    MOREBUFF                YES..NEED MORE BUFFERS
LTR   R15,R15                 RC OTHER THAN Ø?
BNZ   CVAFFERR                YES...DODGY
LA    R3,FCLDEF                R3 -> FCL HEADER
USING FCLMAP,R3                TEMP ADDRESSABILITY TO FCL
   LH   R2,FCLDSCBR             GET DSCB COUNT
   CVD  R2,DWORD                MAKE IT DECIMAL
   UNPK  UNPKFLD(5),DWORD+5(3)  UNPACK IT
   OI   UNPKFLD+4,X'FØ'         SET CORRECT SIGN
   MVC  WTO2+49(1),UNPKFLD+4    WTO2
WTO2  WTO  'WAIT4FTP- NUMBER OF FTP DATASETS LOCATED=?' , ROUTCDE=11
   CLC  FCLDSCBR,=H'3'          GOT ALL 3?
BE    GOTE M                 YES...CARRY ON
WTO3  WTO  'WAIT4FTP- NOT ALL DATASETS HAVE ARRIVED - WAITING...' , X
               ROUTCDE=11
   STIMER WAIT, BINTVL=BIN6Ø   WAIT 6Ø SECONDS
   BCT  R7,CVPLLOOP             HAVE ANOTHER GO
WTO4  WTO  'WAIT4FTP- NOT ALL DATASETS HAVE ARRIVED - TERMINATING' , X
               ROUTCDE=11
   MVC  RETC,F16                SET RC=16
   B     RETURN
GOTE M  DS ØH
OPEN (DSNAMES, OUTPUT) OPEN OUTPUT DCB
LA R3, DSCBDEF POINT TO FIRST DSCB

DSNLOOP DS ØH
MVC OUTREC(44), Ø(R3) MOVE DSNAME TO OUTPUT RECORD
PUT DSNAMES, OUTREC WRITE IT OUT
LA R3, DSCBSIZE(R3) BUMP TO NEXT ONE
BCT R2, DSNLOOP GET THE REST
CLOSE DSNAMES

RETURN DS ØH
L R15, RETC LOAD RETURN CODE
PR , RESTORE CALLER DATA, RETURN

***********************************************************************
* DIDN'T FIND OUR VOLUME...                                          *
***********************************************************************

NOTFND DS ØH
MVC WTO5+26(6), VOLID MOVE VOLID TO WTO
WTO5 WTO 'WAIT4FTP- VOLUME "XXXXXX" NOT LOCATED...', ROUTCDE=11
MVC RETC, F8
B RETURN

***********************************************************************
* INSUFFICIENT BUFFERS TO HOLD DSCBS...                               *
***********************************************************************

MOREBUFF DS ØH
WTO 'WAIT4FTP- TOO MANY DSCBS LOCATED FOR THE # OF BUFFERS DEFINED...', ROUTCDE=11
MVC RETC, F12
B RETURN

***********************************************************************
* BAD RETURN CODE FROM 'UCBSCAN'...                                   *
***********************************************************************

BADCALL DS ØH
LR R1Ø, R15 SAVE RETC
WTO 'WAIT4FTP- BAD CALL TO "UCBSCAN"...', ROUTCDE=11
LR R15, R1Ø RELOAD RETC
DS F BANG

***********************************************************************
* NO/INVALID PARMS PASSED...                                          *
***********************************************************************

BADPARMS DS ØH
WTO 'WAIT4FTP- NO/INVALID PARMS PASSED...', ROUTCDE=11
ABEND 111, DUMP

***********************************************************************
* BAD RETURN CODE FROM 'UCBLOOK'...                                   *
***********************************************************************

LOOKERR DS ØH
LR R1Ø, R15 SAVE RETC
ST RØ, SAVERØ SAVE REASON
WTO 'WAIT4FTP- BAD CALL TO "UCBLOOK"...', ROUTCDE=11
LR RØ, SAVERØ RELOAD REASON
LR R15, R1Ø RELOAD RETC
ABEND 123                      BANG
***********************************************************************
* ERROR IN 'CVAFFILT' MACRO...                                        *
***********************************************************************
CVAFERR DS ØH
LA R1, CVPL+4                TEMP ADDRESSABILITY TO CVAF
USING CVPLMAP, R1
STH R15, HWORD                SAVE RETC
BAL R9, CONVR15               CONVERT RETC TO PRINTABLE HEX
MVC CVAFR15(2), UNPKFLD      MOVE RETC TO DETAIL LINE
MVI HWORD, X'ØØ'              ZEROISE 1ST BYTE
MVC HWORD+1(1), CVSTAT       SAVE CVSTAT STATUS
BAL R9, CONVR15               CONVERT STAT TO PRINTABLE HEX
MVC CVAFSTAT(2), UNPKFLD     MOVE STAT TO DETAIL LINE
MVC WTO6+18(58), CVAFMSG     SHOW CVAFFILT ERROR
WTO6 WTO 'WAIT4FTP: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*
XXXXXXXXXXXXXXXXX', ROUTCDE=11
ABEND 555                      BANG...
DROP R1
***********************************************************************
* SUBROUTINE: CONVERT RETURN CODES, ETC INTO PRINTABLE HEX...          *
***********************************************************************
CONVR15 DS ØH
UNPK UNPKFLD(3), HWORD+1(2)   UNPACK RETC + 1 BYTE
TR UNPKFLD(2), TRTAB-24Ø     XLATE TO PRINTABLE HEX
BR R9                         RETURN FROM ROUTINE
EJECT
*--------------------------------------------------------------------*
*
LTORG                          LITERAL POOL
*
DSNAME    DC    C'FTPXFER.DATA.FILE.PFTP.DNNNTY.**'
DSNSER    EQU    DSNAME+24, 4
DSNAMELN   DC    AL1(*-DSNAME)
VOLID     DC    CL6'??????'
UNPKFLD   DS    XL5
TRTAB     DC    C'0123456789ABCDEF'
HWORD     DC    H'0'
DWORD     DC    D'0'
F4        DC    F'4'
F8        DC    F'8'
F12       DC    F'12'
F16       DC    F'16'
RETC      DC    F'0'
UCBADDR   DC    F'0'
SAVER15   DC    F'0'
SAVERØ    DC    F'0'
SWITCH    DC    C'N'
BIN6Ø     DC    F'1ØØØ'                  6Ø SEC INTERVAL
CVAFMSG   DS    ØCL58
DC CL35'ERROR IN "CVAFFILT" MACRO - RC = X''
CVAFR15 DC CL2' 
   DC CL13' CVSTAT = X''
CVAFSTAT DC CL2' 
   DC CL6''

*---------------------------------------------------------------------*
* UCB SCAN ROUTINE PARAMETERS...                                    *
*---------------------------------------------------------------------*
DS DF
UCCAREA DS XL48                   HOLDS UCB COMMON & DEV SEGS
UCBWORK DS XL100                  UCBSCAN WORKAREA

*---------------------------------------------------------------------*
* DBC FOR OUTPUT...                                                   *
*---------------------------------------------------------------------*
DSNAMES DCB DDNAME=DSNAMES,        X
   LRECL=80,                      X
   RECFM=FB,                      X
   DSORG=PS,                      X
   MACRF=PM
OUTREC DC CL80' 

*---------------------------------------------------------------------*
* SPACE ALLOCATION FOR CVPL, FCL, BFL, AND DSCB BUFFERS...            *
*---------------------------------------------------------------------*
FCLDEF DS (FCLHDLEN+FCLDSNEL)X FCL HEADER AND 1 FCL ELEMENT
FCLIZE EQU *-FCLDEF

*==================================DEFINE A CVAF BUFFER LIST WITH 'N' BUFFER LIST ELEMENTS==================================
BFLHDEF DS (BFLHLEN)X BUFFER LIST HEADER (BFLH)
BFLEDEF DS (BUFFNUM*BFLELN)X 'N' BUFFER LIST ELEMENTS (BFLE'S)
BFLSIZE EQU *-BFLHDEF

*==================================DEFINE 'N' FULL DSCB BUFFERS==================================
DSCBDEF DS (BUFFNUM*DSCBSIZE)X

*--------------------------------------------------------------------*
* REGISTERS EQUATES, ETC...                                           *
*--------------------------------------------------------------------*
BUFFNUM EQU 3                  3 BUFFER LIST ELEMENTS AND BUFFERS

* YREGS

* CVPLMAP I CVAFPL CVPLFSA=YES
FCLMAP I CVFCL
BFLMAP I CVAFBFL
   IEFUCBOB ,
   PRINT NOGEN
   CVT DSECT=YES
DSCBMAP DSECT
   IECSDSL1 (1) FORMAT1 DSCB MAPPING TO BEF BUFFSIZE
DSCBSIZE EQU *-IECSDSL1

* END                     END OF PROGRAM
SAMPLE JCL TO RUN WAIT4FTP

//JOBCARD
//S1       EXEC PGM=WAIT4FTP,PARM='FTP001,1312' <-- VOLID, TIMESTAMP
//STEPLIB   DD DISP=SHR,DSN=TEST.LINKLIB
//SYSPRINT  DD SYSOUT=*  
//DSNAMES   DD DSN=&&DSNAMES,DISP=(MOD,PASS),UNIT=SYSDA,SPACE=(TRK,1)

Grant Carson
Systems Programmer (UK) © Xephon 2003

Data conversion

This utility takes a COBOL copybook and converts it into a REXX include file. If you compile REXX programs, the REXX compiler parses the program looking for special INCLUDE instructions and then copies that code into the source before finishing the compile – much like a pre-processor.

The utility was written to perform data conversion. A project leader in the group decided that we REXX users had to use the same names for our variables as the COBOL people. The COBOL people were getting their copybooks from a data-mapping group, but that left us REXXers out in the cold. Initially, some of us started to manually convert COBOL copybooks to a REXX-like format, but with frequent changes in the data-mapping. Well, you can see that it had problems.

So I wrote the attached utility. With it we did the majority of the conversion using REXX rather than COBOL (much to the chagrin of the aforementioned project leader), and we came in a full five months ahead of schedule.

I have recently found another use for it, and once again it’s a lifesaver.
COPYBOOK REXX

/* ---------------------------- Rexx ---------------------------------*/
/* Title: Convert COBOL Copy Books To REXX Include Format */
/* ===================================================================*/
/* Program Function: */
/* */
/* Converts a COBOL copybook to something REXX can work with. */
/* */
/* Feel free to modify this program as you see fit. */
/* */
/* Syntax: */
/* */
/* COPYBOOK MEMBER SOURCEPDS DESTPDS */
/* */
/* Where: */
/* */
/* COPYBOOK -> This program. */
/* MEMBER -> Name of COBOL copybook member to be converted to REXX. */
/* SOURCEPDS -> PDS containing COBOL copybook members. */
/* DESTPDS -> PDS that will contain the converted copybook in REXX format. */
/* */
/* ==============================================================*/

ConvertCobolCopybooks:
arg member source dest .
call Initialize
call PerformCopyBookConversion
call Finalize
return

/* ------------------------------------------------------------------*/

Initialize:
k = Ø; offset = 1; true = 1; false = Ø ; fieldlen = Ø
level.Ø = ''; variable.Ø = ''; offset.Ø = ''; stackptr = Ø
/* Initialize stacks */
levelq.Ø = ''; variableq.Ø = ''; offsetq.Ø = ''; queuetop = Ø
/* Initialize queues */
lastlevel = Ø
/* Just what it says */
i = Ø
return
/* ------------------------------------------------------------------*/

PerformCopyBookConversion:
call ReadCopybook
do while (i < copybook.Ø)
i = i + 1
copybook = PreProcessCopybook(copybook.i)
parse var copybook level variable pic field remainder
if ((level < lastlevel) | (pic = '' & level = LevelStackTop()))
then
  do while (level <= LevelStackTop())
    call PopStacks
  end
lastlevel = level
select
  when (left(copybook,1) = "*") then
    nop
  when (pic = 'REDEFINES') then
    do
      offset = RedefinesProcessing(field)
      parse var copybook level variable redword redvar
      pic field remainder
      if (field ¬= '') then
        call ProcessCobolLevel
      else
        call PushStacks
      end
    when level = 88 then
      call Level88Processing
    when pic = '' then
      call PushStacks
    otherwise
      call ProcessCobolLevel
    end
  end
end
do while (stackptr ¬= Ø)
call PopStacks
end return
/* ------------------------------------------------------------------*/
PreProcessCopybook:
arg copybook
if (left(copybook,1) ¬= " ") | (left(copybook,1) ¬= ";") then
  parse var copybook 1 junk 7 copybook 73 .
copybook = strip(copybook)
select
  when (copybook = '') then
    copybook = "*
  when (left(copybook,1) ¬= "") then
    copybook = BuildCopybookLine(copybook)
  otherwise
    nop
end return copybook;
/* ------------------------------------------------------------------*/
BuildCopybookLine:
arg line
line = strip(line)
do while ((substr(line,length(line),1) ≠ '.') & (left(line,1) ≠ "*") & (i < copybook.Ø))
  i = i+1; line = line' 'strip(copybook.i)
end
line = substr(line,1,length(line)-1) /* gets rid of period */
return line
/* ------------------------------------------------------------------*/
LevelStackTop:
return level.stackptr
/* ------------------------------------------------------------------*/
VariableStackTop:
return variable.stackptr
/* ------------------------------------------------------------------*/
PushStacks:
  stackptr = stackptr + 1
  variable.stackptr = variable
  level.stackptr = level
  offset.stackptr = offset
return
/* ------------------------------------------------------------------*/
PopStacks:
  newoffset = offset.stackptr
  newlevel = level.stackptr
  newvariable = translate(variable.stackptr,'_','-')
  stackptr = stackptr - 1
  k = k + 1
  outrec.k = " parse var record" left(newoffset,6)
  left(newvariable, 4Ø) offset
  x = Enqueue(newlevel,newvariable,newoffset)
return
/* ------------------------------------------------------------------*/
Level88Processing:
  variable = translate(variable,'_','-')
  field = TranslateField(field)
  k = k + 1
  outrec.k = " /* 88-Level */ "||variable||" = "||lastVariable||"="||field||""
  if (strip(remainder) ≠ "") then
    do
      string = (pos("",remainder) ≠ Ø);
      if (string) then
        do while (remainder ≠ "")
          parse var remainder """val"" remainder
          outrec.k = outrec.k||" |
          ("||lastVariable||"="||val||")
        end
      else
        do while (remainder ≠ "")
  end
```plaintext
parse var remainder val remainder
outrec.k = outrec.k||" |" |
("||lastVariable||"="||val||"|")
end
end
return
/* ------------------------------------------------------------------*/

TranslateField: procedure expose fieldlen
arg field
select
  when (field = 'SPACES') | (field = 'SPACE') then
    field = "" || copies(" ", fieldlen)
  when (field = 'ZERO') | (field = 'ZEROS') then
    field = Ø
  when (field = 'LOW-VALUES') then
    field = copies('ØØ'x, fieldlen)
  when (field = 'HIGH-VALUES') then
    field = copies('FF'x, fieldlen)
  otherwise nop
end
return field
/* ------------------------------------------------------------------*/

ProcessCobolLevel:
value = ''
computational = (pos("COMP-3",remainder) ¬= Ø)
binary = (pos("COMP ",remainder) ¬= Ø)
if pos("VALUE",remainder) ¬= Ø then
  parse var remainder "VALUE" value
fieldlen = DetermineFieldLength(field)
newoffset = fieldlen + offset
k = k + 1
variable = translate(variable, '_', ',' '
lastvariable = variable /* storage bucket in case of 88 levels*/
outrec.k = "  parse var record" left(offset,6) left(variable,4Ø)
newoffset
x = Enqueue(level, variable, offset)
offset = newoffset
if value ¬= '' then
do
  k = k + 1
  outrec.k = variable" = "translateField(value);
end
return
/* ------------------------------------------------------------------*/

RedefinesProcessing: procedure expose variableq. offsetq. queuetop
arg redefinesvariable
queueptr = 1
redefinesvariable = translate(redefinesvariable, '_', ',' '
do while redefinesvariable ¬= variableq.queueptr & queueptr <
```
queueptr = queueptr + 1
end
return offsetq.queueptr
/* -------------------------------------------------------------------*/
Enqueue: procedure expose levelq, variableq, offsetq, queuetop
parse arg level, variable, offset
queuetop = queuetop + 1
levelq.queuetop = level
variableq.queuetop = variable
offsetq.queuetop = offset
return rc
/* -------------------------------------------------------------------*/
DetermineFieldLength: procedure expose computational binary
arg picmap
parse var picmap decimal'V'fraction
fieldlen = 0
if decimal = '' then do
  parse var decimal pre'('len')'
  if len = '' then do
    if substr(pre,1,1) = 'S' then
      len = length(pre) - 1
    else
      len = length(pre)
    end
  end
  fieldlen = fieldlen + len
end
if fraction = '' then do
  parse var fraction pre'('len')'
  if len = '' then
    len = length(pre)
  fieldlen = fieldlen + len
end
select
  when computational then
    fieldlen = trunc((fieldlen + 1)/2)
  when binary then
    select
      when fieldlen <= 4 then fieldlen = 2
      when fieldlen <= 9 then fieldlen = 4
      when fieldlen <= 16 then fieldlen = 6
      otherwise fieldlen = 8
    end
  otherwise nop
end
return fieldlen
/* -------------------------------------------------------------------*/
ReadCopybook:
  source = source'('member')';
  if sysdsn('"source"') != "OK" then
    exit 8
  else
    x = ReadDataset(source, "COPYBOOK")
return
/* ------------------------------- */
ReadDataset:
  parse arg file, stemvar .
  address tso "alloc dd(INDD) da('"file"') shr"
  "execio * diskr INDD (stem "stemvar". finis"
  address tso "free dd(INDD)"
return rc
/* ------------------------------- */
Finalize:
  convertedcopybook = dest'("member")'
  if sysdsn('"dest"') = 'OK' then
    do
      address tso "alloc dd(TEMPDD) da('"dest"') new reuse tr dir(20)",
      "sp(30 30) lrec(255) recfm(V B) dsorg(PO) blksize(0)"
      address tso "free dd(TEMPDD)"
    end
  address tso "alloc dd(OUTDD) da('"convertedcopybook"') shr"
  "execio 'k' diskw OUTDD (stem outrec. finis"
  address tso "free dd(OUTDD)"
return
/* ------------------------------- */

SAMPLE COBOL COPYBOOK
This is an example of a COBOL copybook:

******************************************************************************
* CABT0025 - CUI JURISDICTION TABLE RECORD *
* GTE HISTORY SECTION
* DATE     INITIAL     RVL      DESCRIPTION          DPSR *
* 07/01/87  KAN     R1V01L01   CABS ENHANCEMENT *
* 12/28/88  SAC     R2V01L01   DATE SENSITIVITY CABD0070 *
* 12/28/88  SAC     R2V01L01   FGB MPB INDICATOR CABD0119 *
* 06/05/90  REW     R2V01L01   AT&T 0 EMR1101 CABD0273 *
* 04/28/91  SAC     R2V06L01   DB800 INDICATORS CABD0306 *
* 04/28/91  SAC     R2V06L01   CELL NATL THRESHLD CABD0307 *
* 03/12/93  LB      R1V24L01   ADD VALUE OF 'L' CBSD0838 *
* TO TRAFFIC-TYPE- *
* BILLABLE-INDICATOR *
******************************************************************************

  03 T0025-TABLE-IDENT PIC X(05).
88 T0025-TABLE-ID VALUE 'T0025'.
03 T0025-TABLE-DATA.
05 T0025-KEY.
 10 T0025-PARTIAL-KEY.
   15 T0025-FROM-NPA-NXX.
     20 T0025-FROM-NPA PIC X(03).
     20 T0025-FROM-NXX PIC X(03).
   15 T0025-TO-NPA PIC X(03).
 10 T0025-EFFECTIVE-START PIC 9(6).
 10 T0025-EFFECTIVE-END PIC 9(6).
05 T0025-FUNCTION.
 10 T0025-TABLE-LEVEL PIC X(1).
 10 T0025-JURISDICTION PIC 9.
 10 T0025-BILLING-LOCATION-CODE.
    00002400
 15 T0025-OPERATING-GROUP PIC X(01).
 15 T0025-CONSOL-COMPANY PIC X(01).
 15 T0025-COMPANY PIC X(01).
 15 T0025-AREA-DIV PIC X(01).
 15 T0025-DIV-DIST PIC X(02).
 15 T0025-PLANT-CODE PIC X(04).
 10 T0025-BAN-STATE-CODE PIC X(01).
 10 T0025-ORIG-MMU PIC X(01).
 10 T0025-ORIG-RATE-CNTR PIC 9(03).
 10 T0025-OLATA-CODE PIC 9(03).
 10 T0025-ASSOC-BELL-RAO PIC X(03).
 10 T0025-RCC-THRESHOLD-SEC PIC 9(02).
 10 T0025-IND-EQUAL-ACCESS PIC X(01).
    88 T0025-EQUAL-ACCESS VALUE 'Y'.
    88 T0025-NON-EQUAL-ACCESS VALUE 'N'.
 10 T0025-FGB-MPB-IND PIC X(01).
 10 T0025-FGCD-MPB-IND PIC X(01).
 10 T0025-CONV-OPH-INTRA PIC S9V9(02) COMP-3.
 10 T0025-CONV-OPH-INTER PIC S9V9(02) COMP-3.
 10 T0025-CONV-DIAL PIC S9V9(02) COMP-3.
 10 T0025-RECORD-POINT-DDD PIC S9(06) COMP-3.
 10 T0025-RECORD-POINT-OPH PIC S9(06) COMP-3.
 10 T0025-RECORD-POINT-TERM PIC S9(06) COMP-3.
10 T0025-ATT-SOURCE-OF-DATA.
 15 T0025-ATT-OPH-ZERO-PLUS PIC X(01).
    88 T0025-ATT-TYPE-1-TOLL VALUE 'T' 'L' 'C'
    88 T0025-ATT-TYPE-1-UMS VALUE 'U'.
    88 T0025-ATT-TYPE-1-EMR VALUE 'D'.
 15 T0025-ATT-OPH-ZERO-MINUS PIC X(01).
    88 T0025-ATT-TYPE-2-TOLL VALUE 'T' 'L' 'C'
    88 T0025-ATT-TYPE-2-UMS VALUE 'U'.
    88 T0025-ATT-TYPE-2-EMR VALUE 'D'.
 15 T0025-ATT-MTS-ORIGINAL PIC X(01).
    88 T0025-ATT-TYPE-3-TOLL VALUE 'T' 'L'.
Here is the same sample after it has been run through the utility and converted:
parse var record 1 TØØ25_TABLE_IDENT 6
/* 88-Level */ TØØ25_TABLE_ID = (TØØ25_TABLE_IDENT='TØØ25')
parse var record 6 TØØ25_FROM_NPA 9
parse var record 9 TØØ25_FROM_NXX 12
parse var record 6 TØØ25_FROM_NPA_NXX 12
parse var record 12 TØØ25_TO_NPA 15
parse var record 6 TØØ25_PARTIAL_KEY 15
parse var record 15 TØØ25_EFFECTIVE_START 21
parse var record 21 TØØ25_EFFECTIVE_END 27
parse var record 6 TØØ25_KEY 27
parse var record 27 TØØ25_TABLE_LEVEL 28
parse var record 28 TØØ25_JURISDICTION 29
parse var record 29 TØØ25_STATE_CODE 31
parse var record 31 TØØ25_OPERATING_GROUP 32
parse var record 32 TØØ25_CONSOL_COMPANY 33
parse var record 33 TØØ25_COMPANY 34
parse var record 34 TØØ25_AREA_DIV 35
parse var record 35 TØØ25_DIV_DIST 37
parse var record 37 TØØ25_PLANT_CODE 41
parse var record 41 TØØ25_BILLING_LOCATION_CODE 41
parse var record 42 TØØ25_BAN_STATE_CODE 42
parse var record 43 TØØ25_ORIG_MMU 43
parse var record 44 TØØ25_ORIG_RATE_CNTR 46
parse var record 46 TØØ25_OLATA_CODE 49
parse var record 49 TØØ25_BILLING_RAO 52
parse var record 52 TØØ25_ASSOC_BELL_RAO 55
parse var record 55 TØØ25_RCC_THRESHOLD_SEC 57
parse var record 57 TØØ25_IND_EQUAL_ACCESS 58
/* 88-Level */ TØØ25_EQUAL_ACCESS = (TØØ25_IND_EQUAL_ACCESS='Y')
/* 88-Level */ TØØ25_NON_EQUAL_ACCESS =
(TØØ25_IND_EQUAL_ACCESS='N')
parse var record 58 TØØ25_FGB_MPB_IND 59
parse var record 59 TØØ25_FGCD_MPB_IND 60
parse var record 60 TØØ25_CONV_OPH_INTR A 62
parse var record 62 TØØ25_CONV_OPH_INTER 64
parse var record 64 TØØ25_CONV.Dial 66
parse var record 66 TØØ25_RECORD_POINT_DDD 69
parse var record 69 TØØ25_RECORD_POINT_OPH 72
parse var record 72 TØØ25_RECORD_POINT_TERM 75
parse var record 75 TØØ25_ATT_OPH_ZERO_PLUS 76
/* 88-Level */ TØØ25_ATT_TYPE_1_TOLL =
(TØØ25_ATT_OPH_ZERO_PLUS='T') | (TØØ25_ATT_OPH_ZERO_PLUS='L') |
(TØØ25_ATT_OPH_ZERO_PLUS='C')
/* 88-Level */ TØØ25_ATT_TYPE_1_UMS =
(TØØ25_ATT_OPH_ZERO_PLUS='U')
/* 88-Level */ TØØ25_ATT_TYPE_1_EMR =
(TØØ25_ATT_OPH_ZERO_PLUS='D')
parse var record 76 TØØ25_ATT_OPH_ZERO_MINUS 77
/* 88-Level */ TØØ25_ATT_TYPE_2_TOLL =
(TØØ25_ATT_OPH_ZERO_MINUS='T') | (TØØ25_ATT_OPH_ZERO_MINUS='L') |
Space abend reporter

INTRODUCTION

Since the dawn of computers, every now and then you get given the ‘expert’ advice: “If you prevent application abends, you will improve throughput”. No matter how easy it may sound, doing it for an entire installation is not such a trivial job. The fact is that exponential growth in data storage resulting from data-intensive applications, e-business, and regulatory requirements to retain data have driven the need for more efficient systems to manage
storage resources. It is also a fact that few businesses today can afford processing delays or application failures and reruns.

However, application failures and errors do occur. Although not explicitly responsible for failures in application programs, a storage administrator is the one who is called upon when failures occur, especially if there is even the slightest hint or suspicion that the error is related to storage management. On the other hand, the interpretation of cryptic and often hard to understand IBM operating system error codes and messages can be tedious and frustrating, especially for application programmers. Often the appropriate manuals are not readily available for the users to access. If the manuals are available, the error description and corrective procedure are often impossible to interpret – it is the storage administrator again who must interpret the error and find a cause, and provide the ‘patch’.

Amongst a large variety of possible errors the most common type is the inability to acquire a sufficient amount of DASD space. This usually produces a JCL error if primary space cannot be obtained while an x37 type of error (B37, D37, or E37 abend) is generated if secondary space is unavailable. These abends are simply telling us that the job was not able to obtain enough DASD space to continue. The most frequent response to it is to increase the amount of space requested. Calculating how much would be enough is not an easy task, however, and space requirements are often guessed at – and generally guessed high, to avoid any chance of x37 abends. This can only waste resources and increase volume fragmentation without eliminating x37 abends: a job that requests much more space than it will really use may not only run short of space itself, but may cause errors when the space it holds is unavailable for other jobs. It is often interesting to look at file usage statistics on a system, and one may find some files with up to 90% of their allocated space free. Therefore increasing the amount of space requested may make the situation worse rather than better.

In order to prevent these out-of-space errors, some installations scrutinize applications’ JCL parameters, which turns out to be
not only unreliable but also time-consuming and expensive in terms of human resources.

A more effective and efficient technique is to trap abends, and make available a variety of actions that would allow the job to complete. An important aspect of this automatic ‘only-as-needed’ approach is the assurance that overhead, fragmentation, and other side-effects that waste space are minimized. When a storage administrator develops a management structure and assigns constructs to DFSMS storage groups, the goal is to anticipate and avoid running out of allowable space and abending a job. Even when additional parameters are made available in DFSMS to handle space-related problems, the same issues remain: the complex implementation of global space recovery functions that are processed in an all-or-none fashion. High-use thresholds are established for each storage group in order to leave some percentage of space unused in the storage group to service an unusual space request, or some volumes/storage groups are placed in quiesced status, so that files are allocated only on these volumes/storage groups when a high threshold is exceeded.

Success in reducing the number of $x37$s is directly related to the number of volumes in the storage group – the more volumes each group has, the merrier. The use of dataclass is paramount too: a proper dataclass will ensure extra candidate volumes are allocated to the datasets, so even if the user does allocate insufficient space, DFSMS will prevent it from failing by allowing it to extend to additional volumes. However, if you abuse this multivolume concept, you will have too much waste space, because RLSE is not issued against multivolume datasets. Some users have found this to be a major stumbling block because of the JCL changes necessary to remove the traditional space parameters and to specify the proper dataclass.

Worth the attention of the storage administrator are the space constraint relief enhancements – they do two main things to address out-of-space abends:
• They relax the limit on the number of extents that can be used to satisfy allocations for new datasets or extends to a new volume.
• The amount of space requested can be reduced by a percentage if the initial attempt fails, thus increasing the likelihood of a successful allocation (and, of course, being made up for by subsequent secondary extents).

These are designed to reduce the likelihood of x37 abends, and experience so far shows that they work fairly well.

With z/OS 1.3, DFSMS gets even better at handling out-of-space (x37) abends:
• Extend storage groups – one can specify a second storage group name when defining a storage group. If a dataset wants to extend to another volume, but there is insufficient space in the primary group, the dataset can extend into this storage group. Primary allocations will not use this storage group.
• Overflow storage groups – similar to the above, except that they are used when DFSMS cannot initially allocate onto the primary storage group. This storage group becomes the dataset’s primary storage group, ie second and other volumes must go into this storage group.

One can combine both features so that a storage group can be both an extend and overflow storage group. There is a lot of good information in the redbook z/OS V1R3 DFSMS Technical Guide (SG24-6569).

When it comes to dealing with out-of-space failures for VSAM datasets, things get a bit more complicated. The errors that occur when creating or extending VSAM datasets are believed to be the most cryptic in MVS, mainly consisting of just a return code and a reason code. Among these messages one of the most common is IDC3009I, issued when a catalog function fails. This particular message includes literally hundreds of possible return/reason codes, a few of which are encountered more frequently
than others. A return code 68 says that there is a space error in DADSM (no space is available on the user volume). When accompanied by reason code 20, 22, or 24, it tells us that the storage group did not contain sufficient space to allocate the dataset. More precisely, it means that an attempt to allocate a DFSMS-managed dataset, using the best-fit interface, allocated space on all supplied volumes, but the amount of space allocated was not the total amount required. The DADSM error codes are fully documented in the manual *DFPSMSdfp Diagnosis Reference* (LY27-9606).

**COLLECTING X37 DATA**

No matter how fine-tuned DFSMS might be, the x37 abends will not disappear, simply because DFSMS won’t keep applications from abending – no such facility exists in DFSMS. Obviously, x37 errors can be reduced through the use of DFSMS facilities like extended format datasets or space constraint relief, extend storage groups, and/or overflow storage groups. These are good facilities that can greatly reduce the likelihood, but they don’t stop a job from aborting when allocated space is insufficient.

In order to reduce the occurrence of x37 abends as well as to keep things under control a ‘space abend’ reporting program was written. Running it on a daily basis may help the storage administrator by identifying necessary pool size changes that can be made, and in discussion with the applications programmer on how the dataset got to be bigger than anticipated. This report could also be used to measure and predict how the storage policies impact the user community.

One of the best ways to report on space abends is through the SMF records. Before proceeding any further it might be helpful to understand the context in which certain SMF records, and the values within them, are written. In this particular case, one may be tempted to choose to process less useful SMF type 4 and type 30 records, and report the jobs that abended with B37, D37, and E37 abends.
When enabled by the proper SMFPRMxx TYPE parameter, SMF creates type 42, subtype 9 records, which are written each time an x37 failure occurs for non-VSAM files, while record type 64 (VSAM component/cluster status) was used to report out-of-space abends pertaining to a VSAM dataset. A detailed description of the layout of SMF type 42 and 64 records can be obtained from the *MVS System Management Facilities (SMF), SA22-7630-03* manual. One can also find the type 42 subtype descriptions in macro IGWSMF in SYS1.MACLIB. For type 64 one should consult macro IDASMF64 in the same library.

**CODE**

Based on record descriptions obtained from the manual, a sample report writer was written.

The code is a two-part stream. In the first part (COPYSMF) selected SMF records (selection being defined by INCLUDE conditions) are copied from the SMF dataset to a file, which can be used for archived records.

In the second part (ABD37), the captured records are formatted by invoking EXEC (AB37) and two reports are produced.

The X37 report shows information such as jobname, dataset name, volume serial number, number of dataset extents, DFSMS-related information, as well as date and time. The second report (under DDn V37) is for VSAM out-of-space conditions and it provides information such as jobname, component name, number of extents, etc.

**COPYSMF**

```
//DEL       EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=X
//SYSIN     DD *
    DELETE uid.ABEND.TEST
    DELETE uid.X377.DATA
    DELETE uid.V377.DATA
    SET MAXCC=Ø
```
// COPIYS MF EXEC PGM=I CETOOL
// TOOLMSG DD SYSOUT=* 
// DFSMSG DD SYSOUT=* 
// RAWSMF DD DS N=your.smf.dataset,DISP=SHR 
// SMFX37 DD DS N=ui d. ABEND. TEST, 
// SPACE=(CYL,(1)), UNIT=SYSDA, 
// DIS P=(NEW, CATLG, KEEP), 
// DCB=(RECFM=VB, LRECL=32756, BLKSIZE=32760) 
// TOOLIN DD * 
// COPY FROM(RAWSMF) TO(SMF X37) USING(SMFI) 
// SMFX37 DD DSN=uid.ABEND.TEST, 
// SPACE=(CYL,(1)), UNIT=SYSDA, 
// DISP=(NEW, CATLG, KEEP), 
// DCB=(RECFM=VB, LRECL=32756, BLKSIZE=32760) 
// TOOLIN DD * 
// OPTION SP ANIC=RC4, VLSHR 
// INCLUDE COND=(6,1,B1,EQ,42,AND,23,2,B1,EQ,9, * Get SMF type 42.9 
// OR, (6,1,B1,EQ,64,AND,43,1,B1,EQ,X'20')) * also copy SMF 64 
/* 
// ABD37 EXEC PGM=IKJEFT01, REGION=0M, DYNAMNBR=50, PARM= ' %AB37' 
// SYSEXEC DD DISP=SHR, DSN=your.rexx.library 
// SMF DD DS P=SHR, DSN=uid. ABEND4.TEST 
// X37 DD DS N=uid.X377.DATA, <-- X37 report list 
// SPACE=(CYL,(1,1)), UNIT=SYSDA, 
// DISP=(NEW, KEEP), DCB=(RECFM=FB, LRECL=145) 
// V37 DD DS N=uid.V377.DATA, <-- VSAM report list 
// SPACE=(CYL,(1,1)), UNIT=SYSDA, 
// DISP=(NEW, KEEP), DCB=(RECFM=FB, LRECL=95) 
// SYSPRINT DD SYSOUT=* 
// SYSTSPRT DD SYSOUT=* 
// SYSTSIN DD DUMMY 
/

AB37 EXEC
/* REXX EXEC to read and format space abend records */ 
/* trace ?r */
/*----------------------------------------------------*/
/* Print Space Abend report header and labels */
/*----------------------------------------------------*/

Out. 1 = left(' ',30,')',
    | center('Space Abend Report ',22,)
    | left(' ',15,')

Out. 2 = left(' ',20,' ')
    | center('Report produced on',18,)
    | left(' ',1,')
    | left(date(),11)
    | left('at ',3,')
    | left(time(),10)

Out. 3 = '

Out. 4 = left('Date',11)
    left('Time',9),
    left('Job',6)
    left('Step',4),
    left('X37',4)
    left('Data Set',41),
right('Disp',4)     right('Dsorg',6),
left('Volume',7)    left('Ext.',4),
left('Trks',4)      left('Stor.Cl.',12),
left('Mngt.Cl.',10) left('Data Cl.',10)
Out.5 = LEFT('-',145,'-')

"EXECIO * DISKW X37 (STEM Out.)"

/* Print VSAM 'out-of-space' header and labels */
/*-----------------------------------------------*/
lin.1 = left('',30,' '),
   center('VSAM out-of-space Report ',30),
   left(' ',15,' ')
lin.2 = left('',20,' '),
   center('Report produced on ',18),
   left(' ',1,' ')
   left(date(),11),
   left(' ',1,' ')
   left('at ',3,' ')
   left(time(),10)
lin.3 = ''
lin.4 = left('Date',11) left('Time',9),
   left('Job',8)   left('Component name',45),
   left('Trks',4) right('Ext',4) left('Volume',6)
lin.5 = LEFT('-',145,'-')

"EXECIO * DISKW V37 (STEM Lin.)"

/* Header for SMF records */
/*----------------------------*/
'EXECIO * DISKR SMF ( STEM x. FINIS'
do i = 1 to x.0

smftype  = c2d(SUBSTR(x.i,2,1))             /* SMF record type */
smtime   = smf(c2d(SUBSTR(x.i,3,4)))        /* Decode SMF time */
smfdate  = SUBSTR(c2x(SUBSTR(x.i,7,4)),3,5)  /* Unpack SMF date */
IF smftype = '64' Then call SMF64
IF smftype = '42' Then
   sysid   = SUBSTR(x.i,11,4)           /* System identification */
   sywid   = SUBSTR(x.i,15,4)                    /* Subsystem id */
   smftype = c2d(SUBSTR(x.i,19,2))            /* Record subtype */

/* Product Section */
/*-------------------*/
ops = c2d(SUBSTR(x.i,25,4))       /* Offset to product section */
lps = c2d(SUBSTR(x.i,29,4))       /* Length to product section */

nps = c2d(SUBSTR(x.i,31,4)) /*Number to product sections*/

IF ops <> Ø AND lps <> Ø Then do
  ops = ops - 3
  pdl = SUBSTR(x.i,ops+8,8) /* Product level*/
  pdn = SUBSTR(x.i,ops+8,10) /* Product name*/
  psv = c2d(SUBSTR(x.i,ops+18,1)) /* Subtype version number*/
  SELECT
    when psv = '0' then vhead = 'No vol. header section'
    when psv = '1' then vhead = 'Vol. header exists'
  END
end

/*--------------------------------------------------------------*/
/*   SMF42 subtype 9 header section (B37/D37/E37 abend)         */
/*--------------------------------------------------------------*/
abo = c2d(SUBSTR(x.i,33,4)) /*Offset to X37 abend section */
abl = c2d(SUBSTR(x.i,37,2)) /*Length of X37 abend section */
abn = c2d(SUBSTR(x.i,39,2)) /*Number of X37 abend sections*/
smo = c2d(SUBSTR(x.i,41,4)) /*Offset to SMS data section  */
sml = c2d(SUBSTR(x.i,45,2)) /*Length of SMS data section  */
smn = c2d(SUBSTR(x.i,47,2)) /*Number of SMS data sections */

/*--------------------------------------------------------------*/
/*   B37/D37/E37 Abend Data Section (part 1)                    */
/*--------------------------------------------------------------*/
IF abo <> Ø AND abl <> Ø Then do
  abo = abo - 3
  sys1 = SUBSTR(x.i,abo,4) /* System ID*/
  job = SUBSTR(x.i,abo+4,8) /* Job name*/
  time = smf(c2d(SUBSTR(x.i,abo+12,4))) /*Start time*/
  date = SUBSTR(c2x(SUBSTR(x.i,abo+16,4)),3,5) /*Start date*/
  uid = SUBSTR(x.i,abo+20,8) /* User id.*/
  step = c2d(SUBSTR(x.i,abo+28,1)) /* Step no.*/
  a37b = SUBSTR(x.i,abo+29,1) /* X37 flags*/
  SELECT
    when a37b = '10000000'b then abend = 'B37'
    when a37b = '01000000'b then abend = 'D37'
    when a37b = '00100000'b then abend = 'E37'
  END
  dsorg = c2x(SUBSTR(x.i,abo+34,2)) /*Dataset organization*/
  SELECT
    when dsorg = '4000' then Dorg = 'PS'
    when dsorg = '2000' then Dorg = 'DA'
    when dsorg = '0200' then Dorg = 'PO'
    otherwise dorg = 'PO'
  END

END
disp = SUBSTR(x.i, abo+36, 1) /* Dataset disp*/

SELECT
    when disp = '11000001'b then dis = 'Temp'
    when disp = '11000000'b then dis = 'New'
    when disp = '10000000'b then dis = 'Mod'
    when disp = '01000000'b then dis = 'Old'
    when disp = '01001000'b then dis = 'Shr'
END

dsn = SUBSTR(x.i, abo+37, 44) /* Dataset name*/
vol = SUBSTR(x.i, abo+81, 6) /* Volser of current volume*/
ucb = c2d(SUBSTR(x.i, abo+87, 4)) /* UCB of current volume*/

ext = c2d(SUBSTR(x.i, abo+91, 1)) /* No. of extents for dataset*/
      /* on current volume*/

trk = c2d(SUBSTR(x.i, abo+92, 4)) /* Total tracks allocated for*/
      /* dataset on current volume*/

sal = c2x(SUBSTR(x.i, abo+96, 4)) /* Secondary allocation amount*/

bl = c2d(SUBSTR(x.i, abo+100, 3)) /* Average data block length*/
     /* if specified*/

/* SMS Data Section */

IF smo > 0 Then do
    smo = smo - 3
    mclas = SUBSTR(x.i, smo, 10) /* Management class name*/
    sclas = SUBSTR(x.i, smo+10, 12) /* Storage class name*/
    dclas = SUBSTR(x.i, smo+60, 10) /* Data class name*/
end
else do
    mclas = '
    sclas = 'Non SMS file'
    dclas = '
end

x37out = right(Date('N', date, 'J'), 11) left(sftime, 9),
        left(job, 8) right(step, 2) left(abend, 4),
        left(dsn, 39) right(dis, 6) center(dorg, 6),
        left(vol, 8) right(ext, 3) right(trk, 4),
        right(sclas, 12) right(mclas, 10) right(dclas, 10)

PUSH x37out
"EXECIO 1 DISKW X37"
/* Print abend legend */

Leg.1 = LEFT(' ',145,'-')
Leg.2 = ''
Leg.3 = LEFT('X37 Abend legend:',20)
Leg.4 = ''
Leg.5 = LEFT('B37: All 16 extents used or',60)
Leg.6 = LEFT(' ',5,' ')
   | | | LEFT('No more space was available on the volume or',80)
Leg.7 = LEFT(' ',4,' ')
   | | LEFT('Unable to update the VTOC (full)',78)
Leg.8 = LEFT('D37: Used all the primary space,',33)
   | | LEFT('and no secondary space was requested',40)
Leg.9 = LEFT('E37: Used all space available on',32)
   | | LEFT('the current volume,',31)
Leg.10 = LEFT(' ',5,' ')
   | | | LEFT('and no more volumes were available',40)
Leg.11 = ''
Leg.12 = LEFT('For details see the following messages',70)
Leg.13 = LEFT('B37: IEC030I; D37: IEC031I; E37: IEC032I',80)
"EXECIO * DISKW X37 (STEM Leg.)"
 exit

/* Selected SMF64 variables */

SMF64:
  jbb = SUBSTR(x.i,15,8) /* Job name*/
  rin = SUBSTR(x.i,39,1) /*Recording indicators*/
  SELECT
    when rin='10000000'b then sit='Component closed';
    when rin='01000000'b then sit='Vol switched';
    when rin='00100000'b then sit='No space avail';
    when rin='00010000'b then sit='Cat or CRA rec';
    when rin='00001000'b then sit='Closed type=t ';
    when rin='00001000'b then sit='Abend process ';
    when rin='00000100'b then sit='VVDS or ICF ';
    otherwise sit='Reserved'
  END

  dnm  = SUBSTR(x.i,85,44) /* Dataset name*/
  ntr  = c2d(SUBSTR(x.i,129,2)) /*Number of tracks required*/
  esl  = c2d(SUBSTR(x.i,135,2)) /*Extent segment length*/

/* Extent table - one per volume online at time record written */
noext = esl / 26
/* No. of extent segment */
do extno = 0 to noext - 1
    incr = (140 + (extno * 26)) * 3
    vol.extno = SUBSTR(x.i, incr + 8, 6) /* Volser of curr. volume */
    volume = SUBSTR(x.i, incr + 8, 6) /* Volser of last volume */
    cuu = c2d(substr(x.i, incr + 14, 2)) /* device number */
    ind = substr(x.i, incr + 16, 2) /* Spindle id. */
    uty = c2d(substr(x.i, incr + 18, 4)) /* Unit type - ucbtype */
end

v37out = right(Date('N', smfdate, 'J'), 11) left(smftime, 9),
        left(jbb, 8) left(dnm, 44),
        right(ntr, 4) right(noext, 5) left(volume, 6)

PUSH v37out
    "EXECIO 1 DISKW V37"
return

SMF: procedure
/* REXX - convert a SMF time */
arg time
time1 = time % 100
hh = time1 % 3600
hh = RIGHT('0' || hh, 2)
mm = (time1 % 60) - (hh * 60)
mm = RIGHT('0' || mm, 2)
ss = time1 - (hh * 3600) - (mm * 60)
ss = RIGHT('0' || ss, 2)
otime = hh || ':' || mm || ':' || ss /* Compose SMF time */
return otime

Mile Pekic
Systems Programmer (Serbia and Montenegro) © Xephon 2003

As a free service to subscribers and to remove the need to rekey, the code from individual articles of *MVS Update* can be accessed on our Web site. You will be asked to enter a word from the printed issue.
Extending the life of your mainframe with in-memory table management

In a perfect world, mainframe computing is all about blazing performance, unparalleled scalability, and 99.999% availability. In reality, today’s influx of process/transaction-intensive applications, such as CRM, knowledge management, data mining, Internet-based applications, etc, are taxing mainframe systems. Furthermore, because of budget cutbacks, most companies today are either reluctant or unable to spend extra money on mainframe hardware upgrades that are required to support the increased processing that these large and complex systems necessitate. For many, the answer to this dichotomy is found by making effective use of tables in software applications, coupled with the implementation of an in-memory table management system. For over 20 years, in-memory table management technology has helped Fortune 1000 companies to improve significantly the speed and performance of their mainframe applications, maximize transaction volumes, and support more end-users, without consuming additional hardware resources.

Most of you are aware that using tables in application development means easier and less costly application maintenance because table data, such as a product price list for example, is easily updated without incurring the costs of re-testing the application – and the risk of introducing new errors is virtually eliminated.

Although tables can appear to be a programmer’s best friend, often their processing efficiency is compromised because of the I/O required, especially if the tables are frequently-accessed.

By using an in-memory table management system that loads tables into memory and allows these tables to be shared among applications and regions, application performance is significantly increased while I/O and CPU usage is significantly decreased. This is but one advantage of using an in-memory data management system. There are many others.
An in-memory data management system assures optimal use of existing hardware, and can even go so far as to support the postponement of costly hardware upgrades by making the applications more efficient – the more efficient the application, the less CPU usage is required. In addition, a table management system manages the tables and the memory, unburdening the development and maintenance staff to focus on other tasks.

For those unfamiliar with a table management system, it is similar to a database management system (DBMS), except that tables are loaded into memory. Since data manipulation is performed directly in memory, data access speed is greatly enhanced and the extensive I/O and CPU overhead of disk is avoided. The result is a radical increase in the performance of mainframe applications while extending the useful life of the mainframe system.

Other data management schemes do not provide holistic access to large volumes of structured data. They typically provide only sequential access to small amounts of data – normally thought of as a record, such as customer records from a CRM database or an employee record from an HR application. However, this type of record-level access is not optimal if:

- The data contained in the record needs to be accessed more than once – and possibly repeatedly – during an application run, which means 24x7 for several consecutive weeks for online applications.
- The data needs to be accessed by multiple mainframe applications at the same time, possibly from different regions.
- The data is transient, and only exists for a short period of time, like stock-price quote data.

Data falling into the above categories is handled effectively using a table-driven application design approach coupled with an in-memory table management system.
LOOKING OVER TABLES

Most people accept that a table is an iterative data structure consisting of a series of homogeneous data holders known as rows. Most also realize that a table is a logical structure, independent of its physical representation in memory or on a disk drive.

In the case of programming and database scenarios, the contents of tables can be thought of as consisting of two parts: keys (or indexes) and the data accessed using those keys. When dealing with most catalogue type applications, for example, a part number is used as the key required to access the detailed product description. In many cases, the key occurs within a specific column within the table. See column 1 in the mail-order catalogue example below:

<table>
<thead>
<tr>
<th>Item #</th>
<th>Product</th>
<th>Price</th>
<th>Availability</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP1183</td>
<td>Adobe Photoshop 6.5</td>
<td>549.00</td>
<td>Y</td>
<td><a href="http://www.adobe.com">www.adobe.com</a></td>
</tr>
<tr>
<td>AP1266</td>
<td>Adobe Acrobat Full 5.4</td>
<td>499.00</td>
<td>Y</td>
<td><a href="http://www.adobe.com">www.adobe.com</a></td>
</tr>
<tr>
<td>AP1270</td>
<td>Adobe InDesign 2.0</td>
<td>699.00</td>
<td>N</td>
<td><a href="http://www.adobe.com">www.adobe.com</a></td>
</tr>
<tr>
<td>AP4881</td>
<td>AutoCAD LT 2002</td>
<td>619.95</td>
<td>N</td>
<td><a href="http://www.autodesk.com">www.autodesk.com</a></td>
</tr>
<tr>
<td>AP4895</td>
<td>AutoDesk Symbol 2000</td>
<td>94.95</td>
<td>Y</td>
<td><a href="http://www.autodesk.com">www.autodesk.com</a></td>
</tr>
</tbody>
</table>

The single-key-in-one-column approach may not be flexible or powerful enough in more complex situations. The application may want to use multiple keys – to search for items that are below $500 and are in stock for example – or even use a retrieval mechanism that is not key-centric. In some instances, various applications – and even modules within them – may want to access the data using dissimilar criteria.

Tables, given their inherent logical nature, may be implemented in a number of different ways. Some popular approaches for implementing tables are:

- As structured, and possibly even indexed, files, like VSAM on disk.
- With a relational database using a database management system (DBMS), such as DB2 or Oracle.
• As a programming language array within a specific application, just for use by that application.
• As application-independent data structures resident in memory, external to any specific application programs, that are created, accessed, and maintained using a table management system.

Using memory-based data structures external to applications, which is true in-memory table management, is the only approach that delivers blazing performance, and simplified addition of new data extraction, data manipulation, and data correlation functions to existing applications.

COMBINING DBMSS AND IN-MEMORY TABLE MANAGEMENT

DBMSs and almost all other data-access technologies are designed to work with files, file structures or databases. They operate on the premise that, any time an update is performed, the DBMS will journal that update, reconcile all necessary changes and then make the updated record available to the next user as quickly as possible. Most enterprise data is processed this way. However, this approach is extremely I/O intensive and saps up valuable CPU cycles.

With in-memory table management, these same data tables can be loaded into memory and be readily available. This permits access to that data much faster than DBMS, and ensures that all requisite data manipulations can be performed directly within memory. For heavily-accessed data, application run-times and on-line responses can improve by as much as 90%.

EXTENDING LEGACY TO THE WEB

In-memory table management also helps to support personalization, and is proving particularly useful for extending legacy applications to the Web for real-time, high-volume e-commerce or e-business. It can help accommodate the resulting increase in traffic volumes within existing mainframe CPU, LPAR, and storage constraints, reducing or even eliminating the
need for hardware reallocation or additional hardware resources. Three specific features make table management technology ideally suited for implementing fine-tuned personalization into corporate portal and e-business applications: speed, flexibility, and the capability to develop rules-based applications. All three support a successful delivery of customized user experiences to individual users. For instance, banking institutions that have adopted this technology reportedly credit memory-based table management as the enabling technology for providing personalized statements via the Web. These banks also use table management technology to produce consolidated and personalized master account statements for their customer base. To achieve this, the table-management-centric statement engine application applies 15 to 20 dynamic business rules per customer account in the process of handling millions of transactions per day.

In-memory table management is also suited for implementing incisive rules-based security policies for Web sites and corporate portals, and can be used for interactive, high-volume currency conversion applications as well as sales tax calculation applications. One of Ireland’s largest banking corporations uses an in-memory table management system to handle all of its interactive currency conversion requirements, including the standardization to the euro currency in 2002. The bank’s currency conversion application falls into the repetitive data-access category discussed earlier. Real-time sales tax calculation for point-of-sale applications and online mail-order applications also fall into this repetitive processing category. So do many security-related functions pertaining to corporate portals, whether they are for rules-based security policy enforcement, such as controlling access rights, or data encryption and decryption functions, like those related to SSL security.

BOTTOM LINE
Application design that takes advantage of tables can facilitate a host of requirements, from adding new functionality to existing
applications without requiring any modification or rewrites, to extending mainframe applications to the Web. Combining this approach with an in-memory table management system enables companies to speed-up process-intensive applications, support increased user volumes, optimize system efficiency and performance, facilitate and accelerate application development, and ultimately reduce costs. All in all, in-memory table management is a 20-year-old software methodology that is still in profitable use today, granting a continued lease on life to mission-critical mainframe applications.

Robert Koblovsky  
Vice President of Sales and Marketing  
Data Kinetics Ltd (USA)  

© Robert Koblovsky 2003
BMC Software has announced three new MAINVIEW Storage Resource Manager (SRM) solutions – MAINVIEW SRM Allocation, MAINVIEW SRM Reporting, and MAINVIEW SRM Automation.

These products help customers to manage their mainframe storage environment by reducing storage costs and space shortages, providing automation capabilities that maximize staff and CPU time and minimize wasted space and schedule interruptions.

With these new products, BMC has automated allocation features to reduce the incidence of space-related processing problems and automated actions for the prevention and recovery from abends, ensuring higher levels of availability.

For further information contact: BMC Software, 2101 City West Blvd, Houston, TX 77042, USA. Tel: (713) 918 8800. URL: http://www.bmc.com/products/proddocview/0,2832,19052_19429_6115596_9804,00.html.

Serena Software has announced Version 4.1 of its Application Performance Manager, StarTool.

StarTool APM is a performance measurement and analysis system that helps to resolve OS/390 and z/OS job performance issues. Users can focus their activities on tuning specific areas of an application to boost productivity.

Version 4.1 provides improved, real-time monitoring capabilities with extended support for a wider range of subsystems including CICS, IMS, and DB2.

In addition, Version 4.1 delivers better reporting by providing customers with more in-depth information on an application to help quickly solve performance problems and eliminate the need for further analysis.

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

iWay Software has announced improvements in its data integration capabilities, which include new cluster management, remote installation and management, tighter integration with SQL Server, and new data adapters.

The Remote Installation/Management Console is used for remote installations, which reduces the manual tasks involved in an OS/390, z/OS server installation. Server installations are simplified because the installation files can be transferred and configured from a local laptop or a remote computer, eliminating the need for certain types of technical and operational support.

Intelligent adapters have been added and enhanced to support DB2 Version 8 and IMS Version 8. Additionally, iWay’s Adapter for CICS now supports RRS (Resource Recovery Services) and integrated two-phase commit.

For further information contact: iWay Software, Two Penn Plaza, New York, NY 10121-2898, USA. Tel: (212) 330 1700. URL: http://www.iwaysoftware.com/products/index.html.