December 1997

In this issue

3  Year 2000 – extracting the real-time clock setting
12  A simple search utility
35  The command exit
43  Year 2000 aid: list YEAR2K qualifying records
50  Simulating include files in REXX
62  Organize your disks and claim free space
66  Useful Assembler macros – part 3
72  MVS news

© Xephon plc 1997
INTRODUCTION
In common with many sites, we have logically partitioned our mainframe. Recently one of these partitions was elected to be the official year 2000 test machine. It was decided by the project team to keep the MVS date to a permanent setting (19 January 2000), by re-issuing the MVS SET DATE command daily. This was accomplished by using a JES2 timed command to kick off a started task running batch TSO (IKJEFT01), which would then invoke a REXX routine to issue the MVS SET DATE command via TSO CONSOLE, and then re-set the JES2 timer for the following day (ie in 24 hours’ time).

This worked well. However, keeping the MVS date to a fixed setting has brought its own problems. One of them, which has caused several outages of the partition, is the maintenance of the JES2 queues. On the production partitions we keep only two days’ worth of test job output on the queues. Because we do not have a dedicated SYSOUT archive package, such as SAR, we make use of the JES2 $P Q command, which allows you to specify a number of hours and days. Any output that was created prior to this is purged.

The problem with the year 2000 partition was that all the output had the same date! Therefore, it was impossible to decide which output was old and which was not. It was then agreed that the simplest method would be to clear the queues completely every Sunday night using $OQ,ALL (release all held output), and $PQ,ALL (delete). We could have simply IPLed the machine and performed a JES2 cold start, but we preferred a method we could automate.

Now to the problem. We had a method for issuing a command daily (JES2 timer plus batch TSO). However, because the MVS date was fixed, we had no way of knowing what day of the week it was! If there was a way to get at the real date, then calculating the weekday was relatively simple. However, all date and time functions under REXXX extract the date from MVS. The only way I knew to get at the machine’s real-time clock was from Assembler using the STCK operation (STore ClocK).
Because I needed this information in a REXX routine, I decided to write the SYSDATE() REXX function. This Assembler routine is invoked from a REXX EXEC in the same manner as you would use the built-in DATE() and TIME() functions. However SYSDATE() extracts the machine’s real-time clock value, not the MVS date. The function may be called with two possible arguments:

- The first is with NO ARGUMENTS, ie:

  dat = SYSDATE()

  This will return a string into dat with the format YYYYMMDDHHMMSSHT where:
  - YYYY is the year
  - MM is the month
  - DD is the day
  - HH is hours
  - MM is minutes
  - SS is seconds
  - HT is hundredths and thousandths of a second.

- The second form is:

  day = SYSDATE('W')

  This will return the current day of the week (ie Monday, Tuesday, Wednesday, etc) in the same manner as the REXX DATE('W') function, except that this will be the real weekday.

By using the ‘W’ argument form of SYSDATE, we were able to set up the timed routine to issue only the JES2 queue purge commands on a Sunday. As we expanded the daily timer routines to issue shutdowns for the test CICS regions as well, the first form of SYSDATE became useful for logging purposes.

The source for SYSDATE appears below. The program was developed under MVS5.2 and assembled using High-Level Assembler (ASMA90) Release 1.1.
SOURCE CODE

//jobname JOB 'your job card'
//STEPA EXEC ASMAL.PARM.C='RENT',PARM.L='RENT,REUS'
//C.SYSIN DD *
SYSDATE TITLE 'REXX FUNCTION TO EXTRACT DATE/TIME FROM RTC'

***********************************************************************
*** THIS IS A PROGRAM THAT WILL EXECUTE AS A REXX ***
*** FUNCTION AND WILL RETURN THE DATE/TIME STAMP FROM ***
*** THE MACHINES REAL TIME CLOCK (RTC) INSTEAD OF THE MVS ***
*** DATE/TIME (AS WITH THE STANDARD TSO/REXX DATE() AND TIME() ***
*** FUNCTIONS. ***
***
*** WHEN INVOKED WITH NO PARAMETERS, IE:- ***
*** SDAT = SYSDATE() ***
***
*** THE FUNCTION WILL RETURN A STRING OF THE FORM ***
***
*** YYYYYMMDDHHMMSSHT
*** WHERE:-
***
*** YYYY IS THE CURRENT YEAR
*** MM IS THE CURRENT MONTH
*** DD IS THE CURRENT DAY
*** HH IS THE CURRENT HOUR (24-HOUR FORMAT)
*** MM IS THE CURRENT MINUTE
*** SS IS THE CURRENT SECOND
*** H IS THE CURRENT HUNDREDTH OF A SECOND
*** T IS THE CURRENT THOUSANDTH OF A SECOND
***
***
*** OPTIONALLY, SYSDATE CAN BE INVOKED WITH A SINGLE ARGUMENT ***
*** IF 'W', WHICH WILL RETURN THE CURRENT DAY OF THE WEEK ***
*** IN THE FORM 'MONDAY', 'TUESDAY', ETC. FOR THE CURRENT SYSTEM***
*** DATE. THIS IS EQUIVALENT TO DATE('W') ***
***
*** EG WDAY = SYSDATE('W') ***
***
***********************************************************************
SYSDATE CSECT
SYSDATE AMODE 31
SYSDATE RMODE ANY

BAKR R14,0 *STACK EVERYTHING
LR R12,R15 *R12 --> BASE REGISTER
USING SYSDATE,R12 *ESTABLISH ADDRESSABILITY
LR R10,R0 *R10 --> AENVIRONMENT BLOCK
USING ENVBLOCK,R10 *MAP ENVIRONMENT BLOCK
LR R11,R1 *R11 --> APARAM LIST (EFPL))
USING EFPL,R11 *MAP EFPL
STORAGE OBTAIN.
LENGTH=DYNLEN.
ADDR=(R1), X
LOC=ANY

LR R2,R1    * POINT AT WORKAREA
L R3,-A(DYNLEN) * SET ITS LENGTH
LA R4,0    * SET DUMMY FROM ADDRESS
LA R5,0    * SET DUMMY LENGTH
MVCL R2,R4 * BLANK OUT THE AREA
LR R13,R1 *R13 --)A(DYNAMIC AREA)
USING DYNAM,R13 *ESTABLISH ADDRESSABILITY
L R9,ENVBLOCK_IRXEXTE *R9 --) A(EXTERNAL EP TABLE)
USING IRXEXTE,R9 *MAP IT

***********************************************************************
*** CHECK THE PARAMETER LIST FOR VALID ARGUMENTS ***
*** AND STORE VALUES IN WORKING STORAGE ***
***********************************************************************
*
***********************************************************************
*** FIRST CHECK FOR FUNCTION CODE ***
***********************************************************************

L USING CLC BNE MVI B
TESTARG OS
L CLI BE ell BE B
GOODARG1 OS
MVI ARGFLAG,X'00' * --> NO - CHECK ARG
B GETDATE * --> AND GO GET ..

ARGTABLE_ARGSTRING_PTR(R8),~2F'-1' *END OF ARGS?
BNE TESTARG * --> NO - CHECK ARG
MVI ARGFLAG,X'00' * --> YES - SET FLAG
B GETDATE * --> AND GO GET ..

TESTARG DS ØH
L R2,ARGTABLE_ARGSTRING_PTR *R2 --) A(ARGUMENT)
CLI Ø(R2),X'E6' * UPPERCASE 'W' ?
BE GOODARG1 * YES - CARRY ON
CLI Ø(R2),X'A6' * LOWERCASE 'W' ?
BE GOODARG1 * YES - CARRY ON
B ARGIERR * INVALID FUNCTION

GOODARG1 DS ØH
MVI ARGFLAG,X'01' * SET ARGUMENT FLAG
B GETDATE * GO GET ...

***********************************************************************
*    IF FUNCTION ERROR -    *
* ISSUE ERROR MESSAGE WITH IRXSAY  *
* AND SE RETURN CODE AS 40 TO FLAG INVALID FUNCTION CALL.  *
***********************************************************************

TITLE 'ERROR MESSAGES'
*
ARGIERR DS ØH
LA R1,-C'IRX0000I PARAMETER 1 NOT W OR BLANK'
LA R0,35
B ERROR
*
***********************************************************************
*** SET FUNCTION RESULT ***
***********************************************************************
*
ERROR DS ØH
BAS R14,@SAY * SAY ERROR MESSAGE
*  SET RC=40 TO INDICATE
  INVALID FUNCTION CALL
  *  AND RETURN TO CALLER
*  RETURN

GETDATE DS 0H

*****************************************************************************
*** NOW GET AND FORMAT TIME ***
*****************************************************************************

STCK DWWORK
STCKCONV STCKVAL=DWORK,CONVVAL=OUTAREA,TIMETYPE=DEC,
  DATETYPE=YYYYMMDD,MF=(E,CONVL)
MVC PWORK,PTIME *MOVE TIME TO WORK AREA
MVC PWORK1,-X'0C000000' *MOVE IN PACK CHARACTER
MVO PWORK(9),PWORK *AND OVERLAY TIME
MVC CTIME,-X'F0212020202020202020202020202020202020202020'
ED CTIME,PWORK *FORMAT TIME
MVC PWORK,PDATE *MOVE DATE TO WORK AREA
MVC PWORK1,-X'0C000000' *MOVE IN PACK CHARACTER
MVO PWORK(9),PWORK *AND OVERLAY DATE
MVC CDATE,-X'F0212020202020202020202020202020202020202020'
ED CDATE,PWORK *FORMAT DATE
MVC OUTTIM(8),CDATE+2 *STORE DATE IN MESSAGE
MVC OUTTIM+8(8),CTIME+2 *STORE TIME IN MESSAGE

*  CLI ARGFLAG,X'01'    *  ARGUMENT SPECIFIED?
BE GETDAY    *  GO GET WEEKDAY

*  ELSE ......

*****************************************************************************
*** RETURN FULL DATE ***
*****************************************************************************

L R6,EFPLEVAL *R6 A(-) EVAL BLOCK)
L R6,0(R6) *R6 A(EVAL BLOCK)
USING EVALBLOCK,R6 *MAP EVALBLOCK
  L R15,=F'16'
ST R15,EVALBLOCK_EVLEN *PASS LENGTH OF RESULT
MVC EVALBLOCK_EVDATA(16),OUTTIM *PASS RESULT VALUE
XR R15,R15 *SET RC=0
B RETURN

*****************************************************************************
*** CALCULATE AND RETURN DAY OF WEEK FROM CURRENT DATE ***
*****************************************************************************

GETDAY DS 0H

*  CALCULATE DAY OF WEEK FOR DATE
*  PROGRAM USES A FORMULA KNOWN AS ZELLER'S CONGRUENCE
*  ASSUMING M - MONTH, D - DAY, C - CENTRY NUMBER, Y= YEAR
*  AND THAT 1 - MAR, 2 - APR ... ETC AND THAT
*  JAN AND FEB ARE CLASSED AS MONTHS 11 AND 12 OF THE PREVIOUS YEAR
*  THEN THE FORMULA IS:

  F = (26*M-2)/10 + D + Y + Y/4 + C/4 - 2 * C
* ALL DIVISIONS ARE INTEGER (IE REMAINDERS ARE IGNORED)
* THEN:
* W = F(MOD 7) WILL DENOTE WEEKDAY (0-SUN, 1-MON)
* IF W IS NEGATIVE, ADDING 7 WILL GIVE THE CORRECT NUMBER
* DATE FORMAT = YYYYMMDD
* EXTRACT EACH PARM FROM STORAGE, PACK AND CONVERT TO BINARY FOR
* CALCULATION
STM R14,R12,SAVEAREA *SAVE ALL Registers
LA R3,OUTTIM *ADDRESS DATE
* DAY.....
PACK TEMP(8),6(2,R3) *PACK DAY
CVB R5,TEMP *CONVERT TO BINARY IN R5
ST R5.DAY *AND SAVE (R5 NOW FREE AGAIN)
* MONTH.....
PACK TEMP(8),4(2,R3) *PACK MONTH
CVB R5,TEMP *CONVERT TO BINARY IN R5
* YEAR.....
PACK TEMP(8),0(4,R3) *PACK YEAR
CVB R7,TEMP *CONVERT TO BINARY IN R7
SPACE 2
* NOW DROP 2 FROM MONTH, AND IF NEGATIVE (<0) ADD 12 TO
* ADJUST
S R5,-F'2' *MONTH-2
BP SPLIT *IF >0 GOTO NEXT BIT
A R5,-F'12' *ELSE <0 SO ADD 12 TO ADJUST
BCTR R7,0 *AND DROP 1 FROM YEAR
SPACE 2
* NOW SPLIT YEAR INTO CENTURY AND YEAR BY DIVISION/100
* (CENTURY WILL BE QUOTIENT AND YEAR WILL BE REMAINDER)
SPLIT DS 0H
SR R6,R6 *CLEAR FOR DIVISION
D R6,-F'100' *DIVIDE (R6-YEAR, R7-CENT )
SPACE 2
* AND NOW :
* F = ((26*M-2)/10) + D + Y + Y/4 + C/4 - 2*C
* USING REG 8 AS ACCUMULATOR
SPACE 2
* ((26*M-2)/10) IGNORING REMAINDER ...
M R4,-F'26' * 26*M
S R5,-F'2' * 26*M-2
D R4,-F'10' * (26*M-2)/10
LR R8,R5 * PLACE IN ACCUMULATOR
SPACE 2
* + D + Y - 2*C
A R8,DAY * GET DAY BACK FROM STORE (+D)
AR R8,R6 * +Y
SR R8,R7
SR R8,R7  * - 2*C
SPACE 2
* + Y/4 + C/4
LR R11,R6  * GET Y
SR R10,R10  * BLANK FOR DIVIDE
D R10,-F'4'  * Y/4
AR R8,R11  * ADD TO ACCUM
SR R6,R6  * BLANK FOR DIVIDE
D R6,-F'4'  * C/4
AR R8,R7  * AND ADD TO ACCUM
SPACE 2
* NOW DIVIDE F(MOD7) TO GIVE WEEKDAY NUMBER
SRDL R8,32  * PREPARE FOR DIVIDE (SIGN UNKNOWN)
D R8,-F'7'  * F(MOD 7)
C R8,-F'0'  * <0? (IE NEGATIVE)
BNL *+8  * IF NOT, SKIP NEXT STATEMENT
A R8,-F'7'  * IF NEGATIVE, ADJUST
SPACE 2
* R8 WILL HOLD OFFSET TO TABLE
MH R8,-AL2(9)  * X9 FOR TABLE OFFSET
LA R1,DAYTAB(R8)  * LOAD ADDRESS OF DAY
MVC OUTDAY(9),0(R1)  * MOVE DAY
* NOW ENSURE DAY IN MIXED CASE BY 'OR'ING WITH BLANKS TO FORCE
* TO UPPER CASE, THEN AN EXCLUSIVE OR.
OC OUTDAY,MASK  * FORCE UPPERCASE
XC OUTDAY,MASK1  * AND NOW MIXED CASE
LM R14,R12,SAVEAREA  * RELOAD ALL REGISTERS
**********************************************************************
*** RETURN FULL DATE  ***
**********************************************************************
L R6,EFPLEVAL  * R6 A(-> EVAL BLOCK)
L R6,Ø(R6)  * R6 A(EVAL BLOCK)
USING EVALBLOCK,R6  * MAP EVALBLOCK
L R15,-F'9'
ST R15,EVALBLOCK_EVLEN  * PASS LENGTH OF RESULT
MVC EVALBLOCK_EVDATA(9),OUTDAY  * PASS RESULT VALUE
XR R15,R15  * SET RC-Ø
*
**********************************************************************
*** RETURN TO CALLER  ***
**********************************************************************
RETURN DS ØH
LR R2,R15  * SAVE R15 AROUND RELEASE
STORAGE RELEASE.  * FREE STORAGE BLOCK
LENGTH-DYNLEN,  X
ADDR=(R13)  X
LR R15,R2  * RESTORE RETURN CODE
PR  * RETURN TO CALLER

*** REXX ROUTINE INTERFACES ***
***********************************************************************
* TITLE 'REXX SAY ROUTINE (IRXSAV)'
***********************************************************************
*** INTERFACE TO SAY ROUTINE. ***
*** ON ENTRY: ***
*** R0 - L(BUFFER) ***
*** R1 - A(BUFFER) ***
*** R14 - RETURN ADDRESS ***
*** ***
***********************************************************************

@SAY
ST R14,SAYSAV       *SAVE RETURN ADDRESS
ST R1,SAYP2         *PUT A(RECORD) IN FULLWORD
ST R0,SAYP3         *PASS RECORD LENGTH
LA R0,SAYP1         *INIT PLIST POINTERS
ST R0,SAYPLIST
LA R0,SAYP2
ST R0,SAYPLIST+4
LA R0,SAYP3
ST R0,SAYPLIST+8
LA R0,SAYP4
ST R0,SAYPLIST+12
LA R0,SAYP5
ST R0,SAYPLIST+16
OI SAYPLIST+16,X'80'    *FLAG END OF LIST
MVC SAYPLIST+16,=CL8'WRITE'  *SET FUNCTION
ST R10,SAYP4        *PASS A(ENV BLOCK)
LA R0,FWD           *R0-->A(RETURN CODE AREA)
ST R0,SAYP5         *PASS A(RETURN CODE)

* *
LR R0,R10           *R0-->A(ENV BLOCK)
LA R1,SAYPLIST     *R1-->A(PARAMETER LIST)
L R15,IRXSAY       *R15-->A(USERID ROUTINE)
BALR R14,R15       *ISSUE SAY
LTR R15,R15         *SAY OK?
BZ @SAYOK          *YES
LA R1,-'IRXSAY'    *R1 INDICATE SAY ROUTINE
EX R0,*            *FORCE DIAGNOSTIC ABEND
@SAYOK EQU *
L R14,SAYSAV       *R14--> RETURN ADDRESS
BR R14             *RETURN TO CALLER

***********************************************************************
*** WORKING STORAGE ETC. ***
***********************************************************************

TITLE 'WORKING STORAGE / DSECTS'
MASK DC XL9'404040404040404040'
MASK1 DC XL9'004040404040404040'
DAYTAB DS $H

© 1997. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (940) 455 7050, fax (940) 455 2492.
DC CL9 'SUNDAY'
DC CL9 'MONDAY'
DC CL9 'TUESDAY'
DC CL9 'WEDNESDAY'
DC CL9 'THURSDAY'
DC CL9 'FRIDAY'
DC CL9 'SATURDAY'

LTORG

DYNLEN EQU *-DYNAM

***********************************************************************
*** REQUIRED DSECTS FOR REXX FUNCTIONS ***
***********************************************************************

IRXEFPFL
IRXARGTB
IRXEVALB
IRXENVB
IRXEXTE

***********************************************************************
*** REGISTER EQUATES ***
***********************************************************************

* R0 EQU 0

A simple search utility

Diagnosing problems almost always involves scanning the system log or some other file that contains messages or data. IBM provides a useful utility, ISRSRCHC, that can be invoked under ISPF or executed in batch to search for specific pieces of information. This utility enables you to construct a search consisting of a single string or multiple strings. If you are searching for multiple strings, the utility performs an OR search, if one of the search patterns is found in the current record, the record is output. The ISRSRCHC utility also allows you to search for the occurrence of multiple strings in a record. Both of these search types can be performed in a single execution.

We decided to see if we could construct a similar utility as a programming exercise. The results of our efforts are a routine that we named IEBIBALL. IEBIBALL can perform both the normal OR type
search, where a record will match if it contains one of the search arguments, and an AND search. IEBIBALL can also perform both of these search types in a single execution.

IEBIBALL uses the DSABSERV routine to obtain the dataset names for all of the datasets, as well as to obtain the record type and logical record length for the SYSUT1 dataset.

IEBIBALL is a fairly straightforward utility. There are two key sections of code that you will want to examine. The first section is where the search argument table is constructed. To build the table of search arguments, we first check to see if the first input record is the DELIM= card. IEBIBALL allows you to select the character that you will use as a delimiter. The DELIM= card must be the first input record, or the utility will issue an error message and terminate. If the DELIM= card is present, then the delimiter character is extracted and placed in the translate table. The remainder of the search arguments are then read from the SRCHARGS dataset. Each argument is placed into the search argument table. The length of each argument, as well as a flag which indicates whether AND processing is required, are also placed in the table. The size of the argument table can be adjusted by changing the value of symbol ARG_NUM. The size of the table in the listing that follows is 100 entries. Once the last search argument has been read and processed, the address of the last entry is determined and saved. If the size of the table is exhausted before all of the search arguments are processed, an error message is issued, and the routine terminates. We also check the last search argument to see whether the AND flag is on. This also indicates an error, so we issue a message and terminate the routine.

The second key section of the program is the actual search of each input record. The search of each record is accomplished by using two BXLE loops. The outer BXLE loop is based on the search argument table. The inner BXLE loop is based on the current input record that we are searching for. This is how it works. Registers 9, 10, and 11 access the search argument table. Register 9 has the address of the first entry, register 11 the last entry, and register 10 has the size of each table entry. When we read an input record from the SYSUT1 dataset, we determine whether the file is fixed or variable. If it was fixed, then the LRECL has already been determined for us by the DSABSERV
routine. If it is variable, then the LRECL is extracted from the RDW at the beginning of the record. We use the length of the record, and the length of the search argument, to determine the ending address for the record scan. This ending address is saved. Register 5 is loaded with the beginning address of the current input record. Register 6 is loaded with the scan increment, and register 7 is loaded with the ending address that we have just calculated. Register 5, 6, and 7 comprise the BXLE loop that scans across the input record 1 byte at a time. We use an executable CLC instruction to perform the compare. If we complete the scan BXLE loop and drop through, then the current search argument is not present in the input record. We check to see whether the AND flag is on for the current argument. If it is not, then we adjust register 9 to point at the next search argument, and then go through the process of calculating the ending scan address and perform the scan. If the AND flag was on when we completed the scan, then we manually adjust the contents of register 9 to point to the next search argument. We then check to see if the AND flag is on for this argument. We keep adjusting register 9 in this manner until we do not find the AND flag turned on. When the address in register 9 is greater than the address in register 11, we know that we have searched for all the arguments in the search argument table, and we go to read in a new record from the input file. If we get a match from the compare operation, and the AND flag is turned off for the current search argument, then we output the current record with the record number to the REPORT dataset. If the AND flag is on, then we increment register 9 to point to the next search argument and scan the record again.

IEBIBALL has been assembled and executed under MVS 4.3 and 5.2.2 with DFSMS/MVS 1.3. The files are all coded for 31-bit processing. You can adjust this for 24-bit processing, by modifying the OPEN, CLOSE, and DCB specifications for each of the files. The SYSUT1 dataset can be fixed, variable, or undefined record types. IEBIBALL as coded also supports partitioned datasets in a limited manner. You can point to individual members of a PDS, but you can’t simply point to a PDS and process all the members in a single execution. The program source for DSABSERV has been included, as well as the source for the $ESAPRO, $ESAEPI, $ESASTG and
$CALL macros that were used to develop IEBIBALL. We also executed a few benchmark runs of IEBIBALL and ISRSRCHC against the same input file using the same search arguments. We found that IEBIBALL appears to be more efficient, and on average utilizes about 50% less CPU to obtain the same results. Of course your own results may vary.

SAMPLE JCL TO EXECUTE IEBIBALL

```jcl
//xxxxxxxxx JOB your job card info
//STEPP001 EXEC PGM=IEBIBALL
//STEPLIB DD DISP=SHR,DSN=your.load.library
//SYSABEND DD SYSOUT=* 
//MESSAGES DD SYSOUT=*,DCB=(LRECL=133,RECFM=FBA,BLKSIZE=0)
//REPORT DD SYSOUT=*,DCB=(LRECL=133,RECFM=FBA,BLKSIZE=0)
//SYSUT1 DD DISP=SHR,DSN=file.we.want.search
//SRCHARGS DD *,DCB=(LRECL=80,BLKSIZE=80)
DELIM=+
*TMS001+&
.PRIVAT,+ 
TMS009+
```

IEBIBALL PROGRAM SOURCE

```
TITLE 'IEBIBALL - SCAN UTILITY'

*----+----+----+----+----+----+----+----+----+----+----+----+----+
* CSECT: IEBIBALL
* MODULE: IEBIBALL
* DESC: IEBIBALL IS A SCAN UTILITY SIMILAR TO IBM SEARCH UTILITY
* WHICH IS INVOKED FROM ISPF. IEBIBALL SUPPORTS PHYSICAL
* SEQUENTIAL, PARTITIONED ORGANIZATION, AS WELL AS FILES
* CONTAINING LOAD MODULES. IEBIBALL ALLOWS YOU TO SPECIFY
* A DELIMITER, AS WELL AS SPECIFY THAT YOU WANT ONE
* OR MORE ARGUMENTS TOGETHER. CURRENTLY IEBIBALL WILL
* ACCEPT UP TO 100 SEARCH ARGUMENTS.
* MACROS: $ESAPRO $ESAEP $ESASTG OPEN CLOSE DCB DCBD DCBE
* PUT GET $CALL
* DSECTS: IHADCBD
* INPUT: SYSUT1 - SPECIFIES THE FILE WE WANT TO SEARCH
* SRCHARGS - FILE CONTAINING OUR SEARCH ARGUMENTS
* OUTPUT: MESSAGES - OUTPUT DATASET CONTAINING MESSAGES
* REPORT - OUTPUT FILE LISTING THE RECORDS THAT WERE LOCATED CONTAINING ONE OR MORE OF THE SEARCH
* ARGUMENTS.
* PLIST: NONE
* CALLS: DSABSERV
```

NOTES : 31 BIT ADDRESSING USED FOR ALL FILES.

*----+----+----+----+----+----+----+----+----+----+----+----+----+----*

EJECT

IEBIBALL $ESAPRO R12,AM-31,RM-24

* MAKE SURE THAT WE CAN OPEN UP OUR MESSAGES DATASET. IF NOT WE ARE *
* DONE VERY QUICKLY. *

*----+----+----+----+----+----+----+----+----+----+----+----+----+----*

OPEN (UT3,(OUTPUT)),MODE-31
USING IHADCB,R1
LA R1,UT3
TM DCOFGLS,DCBOFOPN
BO MSG_OPEN

*----+----+----+----+----+----+----+----+----+----+----+----+----+----*

SYN_UT3 DS 0H
MVC RET_CODE,CC_16
B EXIT_RTN

MSG_OPEN DS 0H
MVI UT3_FLAG,DCBOFOPN

*----+----+----+----+----+----+----+----+----+----+----+----+----+----*

LOAD DSABSERV INTO VIRTUAL STORAGE AND SAVE THE ENTRY POINT ADDRESS.*

LOAD EP-DSABSERV,ERRET-LOAD_ERR
B LOAD_OK

LOAD_ERR DS 0H
MVI O_LINE,C' '
MVC O_LINE+1(L'O LINE-1),O_LINE
MVC O_LINE(EM_001L),EM_001
PUT UT3,O_LINE
MVC RET_CODE,CC_16
B EXIT_RTN

LOAD_OK DS 0H
ST R0,@DSAB

*----+----+----+----+----+----+----+----+----+----+----+----+----+----*

OPEN (UT4,(INPUT)),MODE-31
LA R1,UT4
TM DCOFGLS,DCBOFOPN
BO ARG_OPEN

*----+----+----+----+----+----+----+----+----+----+----+----+----+----*

SYN_UT4 DS 0H
MVI O_LINE,C' '
MVC O_LINE+1(L'O LINE-1),O_LINE
MVC O_LINE(EM_002L),EM_002 MOVE IN THE MESSAGE
PUT UT3,O_LINE
MVC RET_CODE,CC_16 SET THE RETURN CODE
B EXIT_RTN GO CLOSE MESSAGES FILE
ARG_OPEN DS @H
MVI UT4_FLAG,DCBOFOPN INDICATE THE DATASET IS OPEN
* OPEN THE REPORT FILE. *
*-----------------------------------------------*
OPEN (UT2,(OUTPUT)),MODE=31
LA R1,UT2 GET @(DCB WE JUST OPENED)
TM DCBOFLGS,DCBOFOPN Q. OPEN SUCCESSFULL ?
BO UT2_OPEN A. YES
* SYNAD CONTROL POINT FOR PHYSICAL ERROR ON THE UT2 DATASET. *
*-----------------------------------------------*
SYN_UT2 DS @H
MVI O_LINE,C' ' PUT BLANK IN BYTE ONE
MVC O_LINE+1(L'O_LINE-1),O_LINE BLANK OUT REMAINDER
MVC O_LINE(EM_003L),EM_003 MOVE IN THE MESSAGE
PUT UT3,O_LINE
MVC RET_CODE,CC_16 SET THE RETURN CODE
B EXIT_RTN GO TO COMMON EXIT POINT
* OPEN THE FILE THAT WE WANT TO SEARCH THROUGH. *
*-----------------------------------------------*
UT2_OPEN DS @H
MVI UT2_FLAG,DCBOFOPN INDICATE DATASET IS OPEN
OPEN (UT1,(INPUT)),MODE=31
LA R1,UT1 GET @(DCB WE JUST OPENED)
TM DCBOFLGS,DCBOFOPN Q. OPEN SUCCESSFULL ?
BO UT1_OPEN A. YES
* SYNAD CONTROL POINT FOR PHYSICAL ERROR ON THE UT1 DATASET. *
*-----------------------------------------------*
SYN_UT1 DS @H
MVI O_LINE,C' ' PUT BLANK IN BYTE ONE
MVC O_LINE+1(L'O_LINE-1),O_LINE BLANK OUT REMAINDER
MVC O_LINE(EM_004L),EM_004 MOVE IN THE MESSAGE
PUT UT3,O_LINE
MVC RET_CODE,CC_16 SET THE RETURN CODE
B EXIT_RTN GO TO COMMON EXIT POINT
UT1_OPEN DS @H
MVI UT1_FLAG,DCBOFOPN INDICATE DATASET IS OPEN
* CALL THE DSABSERV ROUTINE. WE WILL PASS A SET OF QUINTUPLETS TO THE* 
* ROUTINE. EACH QUINTUPLET CONSISTING OF THE FOLLOWING: *
* ADDRESS(HALFWORD FOR THE LENGTH OF THE DATASET NAME) *
* ADDRESS(8 BYTE ARE WITH THE DDNAME WE ARE INTERESTED IN) *
* ADDRESS(44 BYTE AREA FOR THE RETURNED DATASET NAME *)
* WILL CONTAIN 44 ASTERISKS IF DSABSERV WAS NOT ABLE *
TO OBTAIN THE DATASET NAME.)
ADDRESS(LOGICAL RECORD LENGTH, DATASET ORGNIZATION)
ADDRESS(RECORD FORMAT, FIXED OR VARIABLE)

CALL @DSAB,(UT1_L, UT1_DDN, UT1_DSN, UT1_LREC, UT1_RT,

+ UT2_L, UT2_DDN, UT2_DSN, UT2_LREC, UT2_RT,

+ UT3_L, UT3_DDN, UT3_DSN, UT3_LREC, UT3_RT,

+ UT4_L, UT4_DDN, UT4_DSN, UT4_LREC, UT4_RT),

VL, BM=BASSM, MF=(E, PLIST)

TO OUTPUT INFORMATION ABOUT EACH OF THE FILES THAT WE HAVE OPENED.

MVI O_LINE, C'
MVC O_LINE+1(L'O_LINE-1), O_LINE BLANK OUT REMAINDER
MVC O_LINE(OP_001L), OP_001 MOVE IN THE MESSAGE
MVC O_LINE+OP_001D(L'UT1_DSN), UT1_DSN MOVE IN DSNAME
MVC O_LINE+OP_001C(L'UT1_DSO), UT1_DSO MOVE IN DSORG
MVC O_LINE+OP_001E(L'UT1_RT), UT1_RT MOVE IN RECORD TYPE
LH R14, UT1_LREC GET LOGICAL RECORD LENGTH
CVD R14, D_WORK CONVERT IT TO DECIMAL
UNPK O_LINE+OP_001F(5), D_WORK+5(3) UNPACK IT
OI O_LINE+OP_001F+4, X'F0' FIX THE SIGN
PUT UT3, O_LINE

MVI O_LINE, C'
MVC O_LINE+1(L'O_LINE-1), O_LINE BLANK OUT REMAINDER
MVC O_LINE(OP_002L), OP_002 MOVE IN THE MESSAGE
MVC O_LINE+OP_002D(L'UT2_DSN), UT2_DSN MOVE IN DSNAME
MVC O_LINE+OP_002C(L'UT2_DSO), UT2_DSO MOVE IN DSORG
MVC O_LINE+OP_002E(L'UT2_RT), UT2_RT MOVE IN RECORD TYPE
LH R14, UT2_LREC GET LOGICAL RECORD LENGTH
CVD R14, D_WORK CONVERT IT TO DECIMAL
UNPK O_LINE+OP_002F(5), D_WORK+5(3) UNPACK IT
OI O_LINE+OP_002F+4, X'F0' FIX THE SIGN
PUT UT3, O_LINE

MVI O_LINE, C'
MVC O_LINE+1(L'O_LINE-1), O_LINE BLANK OUT REMAINDER
MVC O_LINE(OP_003L), OP_003 MOVE IN THE MESSAGE
MVC O_LINE+OP_003D(L'UT3_DSN), UT3_DSN MOVE IN DSNAME
MVC O_LINE+OP_003C(L'UT3_DSO), UT3_DSO MOVE IN DSORG
MVC O_LINE+OP_003E(L'UT3_RT), UT3_RT MOVE IN RECORD TYPE
LH R14, UT3_LREC GET LOGICAL RECORD LENGTH
CVD R14, D_WORK CONVERT IT TO DECIMAL
UNPK O_LINE+OP_003F(5), D_WORK+5(3) UNPACK IT
OI O_LINE+OP_003F+4, X'F0' FIX THE SIGN
PUT UT3, O_LINE

MVI O_LINE, C'
MVC O_LINE+1(L'O_LINE-1), O_LINE BLANK OUT REMAINDER
MVC O_LINE(OP_004L), OP_004 MOVE IN THE MESSAGE
MVC O_LINE+OP_004D(L'UT4_DSN), UT4_DSN MOVE IN DSNAME
MVC O_LINE+OP_004C(L'UT4_DSO), UT4_DSO MOVE IN DSORG
MVC O_LINE+OP_004E(L'UT4_RT), UT4_RT MOVE IN RECORD TYPE
LH R14,UT4_LREC  GET LOGICAL RECORD LENGTH
CVD R14,D_WORK  CONVERT IT TO DECIMAL
UNPK 0_LINE+OP_04F(5),D_WORK+5(3) UNPACK IT
OI 0_LINE+OP_04F+4,X'F0' FIX THE SIGN
PUT UT3,0_LINE

* AT THIS POINT WE READ IN THE FIRST RECORD FROM THE SRCHARGS FILE *
* WHICH IS POINTED TO BY THE UT4 DCB. THE FIRST RECORD MUST CONTAIN *
* THE DELIM- IN CARD COLUMN 1. IF IT DOES NOT, THEN THE ROUTINE WILL *
* ISSUE AN ERROR MESSAGE, AND TERMINATE. *

GET UT4
LR R2,R1  GET @(CURRENT RECORD)
CLC DELIM,0(R2)  Q. FIRST CARD THE DELIM CARD?
BE GOT_DELM  A. YES, WE CAN PROCEED
MVI O_LINE,' ' PUT BLANK IN BYTE ONE
MVC O_LINE+I(L'O_LINE-I),O_LINE BLANK OUT REMAINDER
MVC O_LINE(EM_13135L),EM_13135 MOVE IN THE MESSAGE
PUT UT3,0_LINE
MVC RET_CODE,CC_16 SET THE RETURN CODE
B EXIT_RTN GO TO COMMON EXIT POINT

* WE HAVE A DELIMITER. PICK IT UP AND POPULATE IT INTO OUR TRANSLATE *
* TABLE. *

GOT_DELM DS 0H
XR R3,R3  CLEAR REG 3
IC R3,L'DELIM(R2)  GET THE DELIMITER
LA R4,TRAN_TAB GET @(TRANSLATE TABLE)
STC R3,0(R3,R4) PLACE CHARACTER IN THE TABLE

* PICK UP THE NEEDED INFORMATION FOR THE BXLE LOOP THAT WILL BE USED *
* TO POPULATE THE SEARCH ARGUMENT TABLE. *

LA R3,ARG_L  GET @(FIRST ENTRY)
ST R3,ARG_TB SAVE IT FOR BXLE
L R4,ARG_LE GET DISPLACEMENT
LA R3,0(R4,R3) CALC @(LAST ENTRY)
ST R3,ARG_TE SAVE IT FOR BXLE
LA R3,ARG_ENTL GET SIZE OF EACH ENTRY
ST R3,ARG_TI SAVE IT FOR BXLE
LM R7,R9,ARG_TB LOAD REGS FOR BXLE LOOP

* READ THE REMAINDER OF RECORDS FROM THE SRCHARGS FILE. EACH ENTRY *
* IS CHECKED TO DETERMINE IF IT END WITH A VALID DELIMITER. WE CHECK *
* FOR THE DELIMITER BY EXECUTING A TRT INSTRUCTION. IF THE RECORD *
* DOES NOT TERMINATE WITH A VALID DELIMITER, WE ISSUE A MESSAGE TO THE*
* MESSAGES DATASET, AND PROCESSING CONTINUES. *

LOOP_UT4 DS 0H
GET UT4
LR R3,R1  GET @(RECORD JUST READ)
LH R14,UT4_LREC  GET THE RECORD LENGTH
BCTR R14,Ø  DECREMENT THE LENGTH
EX R14,TRT_I  Q. DELIMITER LOCATED.
BC 8.ERR_DLM  A. DELIMITER NOT LOCATED
BC 4,CALC_LEN  A. FOUND THE DELIMITER
B LOOP_UT4  SHOULD NEVER GET HERE

ERR_DLM DS ØH
MVI 0_LINE,C' '  PUT BLANK IN BYTE ONE
MVC 0_LINE+1(L'O_LINE-1),0_LINE BLANK OUT REMAINDER
MVC 0_LINE+(EM_007L),EM_007 MOVE IN THE MESSAGE
MVC 0_LINE+EM_007D(80),Ø(R3) COPY SEARCH ARGUMENT
PUT UT3,0_LINE
B LOOP_UT4  READ ANOTHER SEARCH ARG

CALC_LEN DS ØH
LR R14,R1  PICK UP WHERE R1 IS --->>
SR R14,R3  CALCULATE ARG LENGTH - 1
BCTR R14,0(R7)  DECREMENT IT BY 1
STH R14,0(R7)  SAVE THE LENGTH
MVI AND_FLAG-ARG_L(R7),AND_OFF TURN THE AND FLAG ON
EX R14,MVC_I  MOVE THE ARGUMENT
LA R3,(,R1)  BUMP THE ADDRESS
CLI Ø(R3),'X'50'  Q. USER WANT TO AND WITH NEXT
BNE BXLE_GO  A. NO
MVI AND_FLAG-ARG_L(R7),AND_ON TURN THE AND FLAG ON

BXLE_GO DS ØH
BXLE R7,R8,LOOP_UT4  GO GET ANOTHER ENTRY
MVI 0_LINE,C' '  PUT BLANK IN BYTE ONE
MVC 0_LINE+1(L'O_LINE-1),0_LINE BLANK OUT REMAINDER
MVC 0_LINE+(EM_006L),EM_006 MOVE IN THE MESSAGE
PUT UT3,0_LINE
MVC RET_CODE,CC16  SET THE RETURN CODE
B EXIT_RTN  GO TO COMMON EXIT POINT

SIDE_EOF_DS 0H
SR R7,R8  BUMP DOWN TO LAST ENTRY
CLI AND_FLAG-ARG_L(R7),AND_ON Q. IS THE AND FLAG ON
BNE AND_OFF  A. NO, AND FLAG IS OFF
MVI 0_LINE,C' '  PUT BLANK IN BYTE ONE
MVC 0_LINE+1(L'O_LINE-1),0_LINE BLANK OUT REMAINDER

© 1997. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (940) 455 7050, fax (940) 455 2492.
MVC O_LINE(EM_008L),EM_008  MOVE IN THE MESSAGE
PUT UT3,O_LINE
B EXIT_RTN  EXIT THE PROGRAM

AND_OFFF DS ØH
ST R7,ARG_TE  SAVE AS LAST ENTRY
XC UT4_FLAG,UT4_FLAG  CLEAR FLAG BYTE
LA R2,1  PRIME R2
ST R2,R_BXLE+4  SAVE IN SCAN BXLE AREA
ZAP RECORD_R,PACK_0  ZERO OUT RECORD NUMBER
ZAP RECORD_M,PACK_0  ZERO OUT RECORD NUMBER
ZAP RECORD_N,PACK_0  ZERO OUT RECORD NUMBER

*-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----*
* THE SEARCH IS ACCOMPLISHED BY USING A PAIR OF BXLE LOOPS. THE OUTER *
* LOOP IS USED TO PROCESS THE SEARCH ARGUMENT TABLE. R9 POINTS AT THE *
* CURRENT ENTRY, R10 CONTAINS THE INCREMENT, AND R11 POINTS AT THE *
* LAST SEARCH ARGUMENT IN THE TABLE. THE INNER BXLE LOOP IS USED TO *
* SCAN ACROSS THE CURRENT RECORD. R5 POINTS AT THE CURRENT BYTE LO- *
* CATION IN THE RECORD. R6 CONTAINS THE INCREMENT, IN THIS CASE 1, *
* AND R7 CONTAINS THE END POINT IN THE BUFFER. THE END POINT FOR EACH *
* RECORD IS CALCULATED BY TAKING THE SIZE OF THE RECORD, AND SUB-RAC *
* TRACTING OFF THE LENGTH OF THE CURRENT ARGUMENT. *
*-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----*

LOOP_UTI DS ØH
GET UTI
LM R9,R11,ARG_TB  GET TABLE INFO
AP RECORD_R,PACK_1  BUMP RECORD READ COUNTER
AP RECORD_N,PACK_1  INCREMENT CURRENT RECORD #
PRIME_R2 DS ØH
LR R2,R1  GET @(RECORD JUST READ)
CLI UT1_RT,LRECL_F  Q. FIXED RECORD
BNE VAR_UT1  A. NO, DO VARIABLE WORK
ST R2,R_BXLE  SAVE BEGINNING ADDRESS
LH R3,UT1_LREC  GET LOGICAL RECORD LENGTH
B COM_UT1  BRANCH TO COMMON CODE

VAR_UT1 DS ØH
LH R3,0(R2)  GET THE CURRENT RECORD LENGTH
SH R3,HALF_4  ACCOUNT FOR THE RDW
LA R2,4(R2)  ACCOUNT FOR THE RDW
ST R2,R_BXLE  SAVE BEGINNING ADDRESS

COM_UT1 DS ØH
SH R3,0(R9)  SUBTRACT LENGTH OF ARGUMENT
BCTR R3,0  DECREMENT BY ONE
LA R2,0(R3,R2)  CALCULATE ENDING ADDRESS
ST R2,R_BXLE+8  SAVE ENDING ADDRESS
LM R5,R7,R_BXLE  PRIME FOR SCAN LOOP
LH R2,0(R9)  GET LENGTH OF ARGUMENT

*-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----*
* PERFORM THE COMPARE. WE DO THIS BY EXECUTING A CLC. R2 HAS THE *
* LENGTH OF THE CURRENT SEARCH ARGUMENT. *
*-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----*

SCAN_GO DS ØH

EX R2, CLC_I  Q. PATTERN MATCH
BNE NO_MATCH  A. NO,
CLI AND_FLAG-ARG_L(R9), AND_ON Q. AND FLAG ON ??
BE BXLE_BU
B AND_NON  GO OUTPUT THE RECORD

NO_MATCH DS ØH
BXLE R5, R6, SCAN_GO  KEEP SCANNING RECORD

* ---+----+----+----+----+----+----+----+----+----+----+----+----+----*
* IF WE GET HERE, WE HAVE SCANNED THE ENTIRE RECORD AND DID NOT FIND *
* A MATCH. SEE IF THE AND FLAG WAS ON FOR THE CURRENT ARGUMENT. IF *
* IT IS, MANUALLY BUMP R9 UNTIL WE DON'T FIND THE AND FLAG ON. *
*----+----+----+----+----+----+----+----+----+----+----+----+----+----*

MAN_R9 DS ØH
CLI AND_FLAG-ARG_L(R9), AND_ON Q. AND FLAG ON
BNE BXLE_BU  A. NO, GET NEXT SEARCH ARGUMENT
LA R9, Ø(R10, R9)  MANUALLY ADJUST R9
B MAN_R9  GO TEST NEXT ARG

BXLE_BU DS ØH
BXLE R9, R10, PRIME_R2  START SCAN AGAIN
B LOOP_UT1  GO GET NEXT RECORD

* ---+----+----+----+----+----+----+----+----+----+----+----+----+----*
* WE HAVE FOUND A SEARCH ARGUMENT. OUTPUT THE CURRENT RECORD. *
*----+----+----+----+----+----+----+----+----+----+----+----+----+----*

AND_NON DS ØH
AP RECORD_M, PACK_1  INCREMENT THE MATCH COUNTER
MVI O_LINE, C' '  PUT BLANK IN BYTE ONE
MVC O_LINE+1(L'O_LINE-1), O_LINE BLANK OUT REMAINDER
MVC O_LINE(OP_MSG5), OP_MSG5  MOVE IN THE MESSAGE
LA R5, MAX_5  GET MAX ALLOWABLE
CLI UT1_RT, LRECL_F  Q. FIXED RECORD
BNE UT1_VF  A. NO, VARIABLE
LH R6, UT1_LREC  GET ACTUAL RECORD SIZE
CR R6, R5  COMPARE TO THE MAX ALLOWABLE
BNH REC_MOVR  GO MOVE THE RECORD TO BUFFER
LR R6, R5  SET R6 TO THE MAX
B REC_MOVR  MOVE THE RECORD

UT1_VF DS ØH
LH R6, Ø(R1)  GET LENGTH FROM THE RDW
LA R1, 4(R1)  BUMP PAST THE RDW
CR R6, R5  COMPARE TO THE MAX ALLOWABLE
BNH REC_MOVR  GO MOVE THE RECORD TO BUFFER

REC_MOVR DS ØH
EX R6, MVC_RR  MOVE THE RECORD TO O_LINE
UNPK O_LINE+1(6), RECORD_N(4) UNPACK RECORD NUMBER
OI O_LINE+6, X'F0'  FIX THE SIGN
PUT UT2, O_LINE
B LOOP_UT1  GET NEXT RECORD

EOF_UT1 DS ØH
XC UT1_FLAG, UT1_FLAG  CLEAR FLAG BYTE
MVI O_LINE, C' '  PUT BLANK IN BYTE ONE
MVC O_LINE+1(L'O_LINE-1), O_LINE BLANK OUT REMAINDER
MVC O_LINE(OP_006L),OP_006 MOVE IN THE MESSAGE
UNPK O_LINE+1(8),RECORD_R(6) UNPACK RECORD NUMBER
OI O_LINE+8,X'F0' FIX THE SIGN
PUT UT3,O_LINE
MVI O_LINE,C' ' PUT BLANK IN BYTE ONE
MVC O_LINE+1(L'O_LINE-1),O_LINE BLANK OUT REMAINDER
MVC O_LINE(OP_007L),OP_007 MOVE IN THE MESSAGE
UNPK O_LINE+1(6),RECORD_M(4) UNPACK RECORD NUMBER
OI O_LINE+6,X'F0' FIX THE SIGN
PUT UT3,O_LINE
B EXIT_RTN EXIT THE ROUTINE

* COMMON EXIT POINT. CLOSE FILES AS NEEDED AND EXIT. *

EXIT_RTN DS ØH
TM UT1_FLAG,DCBOFOPN Q. DATASET OPEN
BNO UT1_XXX A. NO, CHECK NEXT DATASET
CLOSE (UT1),MODE=31

UT1_XXX DS ØH
TM UT2_FLAG,DCBOFOPN Q. DATASET OPEN
BNO UT2_XXX A. NO, CHECK NEXT DATASET
CLOSE (UT2),MODE=31

UT2_XXX DS ØH
TM UT3_FLAG,DCBOFOPN Q. DATASET OPEN
BNO UT3_XXX A. NO, CHECK NEXT DATASET
CLOSE (UT3),MODE=31

UT3_XXX DS ØH
TM UT4_FLAG,DCBOFOPN Q. DATASET OPEN
BNO UT4_XXX A. NO, ALL DONE
CLOSE (UT4),MODE=31

UT4_XXX DS ØH
$ESAEP RET_CODE
TITLE 'IEBIBALL - LITERALS AND CONSTANTS'
LRECL_F EQU C'F' USED FOR RECORD TYPE TESTING
AND_ON EQU C'Y' USED FOR AND PROCESSING
AND_OFF EQU C'N' USED FOR AND PROCESSING
MVC_RR MVC O_LINE+OP_005L(*-*),O(R1) EXECUTABLE MOVE
MVC_I MVC 3(*-*),R7),O(R3) EXECUTABLE MOVE
CLC_I CLC 0(*-*),R5),3(R9) EXECUTABLE COMPARE
TRT_I TRT 0(*-*),R3),TRAN.Tab FIND THE DELIMITER
ARG_LE DC A(ARG_NUM*ARG_ENTL) DISPLACEMENT TO LAST ENTRY
CC_16 DC F'16' USED TO SET A RETURN CODE
HALF_4 DC H'4' USED FOR RDW ADJUSTMENT
PACK_0 DC PL4'0' USED TO PRIME FIELDS
PACK_1 DC PL4'1' USED TO INCREMENT COUNTERS
DELIM DC CL60'\Delim=' TITLE 'IEBIBALL - MESSAGES'
NO_MSG DC H'60'
DC CL60'UNABLE TO OPEN THE MESSAGES FILE - EXECUTION TERMINATED'
EM_001 DC C'A ERROR HAS OCCURRED TRYING TO LOCATE AND LOAD THE
DSABSERV ROUTINE. IEBIBALL TERMINATING

EM_001L EQU *-EM_001 LET THE ASSEMBLER CALC LENGTH
EM_002 DC C'A ERROR HAS OCCURRED WHILE TRYING TO OPEN THE SEARCH ARGUMENTS DATASET. IEBIBALL TERMINATING'
EM_002L EQU *-EM_002 LET THE ASSEMBLER CALC LENGTH
EM_003 DC C'A ERROR HAS OCCURRED WHILE TRYING TO OPEN THE REPORT DATASET. IEBIBALL TERMINATING'
EM_003L EQU *-EM_003 LET THE ASSEMBLER CALC LENGTH
EM_004 DC C'A ERROR HAS OCCURRED WHILE TRYING TO OPEN THE SYSUTI DATASET. IEBIBALL TERMINATING'
EM_004L EQU *-EM_004 LET THE ASSEMBLER CALC LENGTH
EM_005 DC C'FIRST CARD ENCOUNTERED IN SEARCH ARGUMENTS WAS NOT THE+ DELIM- CARD. IEBIBALL TERMINATING'
EM_005L EQU *-EM_005 LET THE ASSEMBLER CALC LENGTH
EM_006 DC C'MORE THAN 100 SEARCH ARGUMENTS ENCOUNTERED. IEBIBALL TERMINATING.'
EM_006L EQU *-EM_006 LET THE ASSEMBLER CALC LENGTH
EM_007 DC C' MISSING DELIMITER. CARD IMAGE='
EM_007D EQU *-EM_007 LET THE ASSEMBLER CALC LENGTH
DC CLB0' '
EM_007L EQU *-EM_007 LET THE ASSEMBLER CALC LENGTH
EM_008 DC C' AND OPERATION REQUESTED ON THE LAST SEARCH ARGUMENT. + IEBIBALL TERMINATING'
EM_008L EQU *-EM_008 LET THE ASSEMBLER CALC LENGTH
OP_001 DC C' SYSUTI DSNAME='
OP_001D EQU *-OP_001 LET ASSEMBLER CALCULATE LENGTH
DC CL44' '
DC C' DSORG='
OP_001C EQU *-OP_001
DC CL2' '
DC C' RECFM='
OP_001E EQU *-OP_001
DC CL2' '
DC C' LRECL='
OP_001F EQU *-OP_001
DC CL5' '
OP_001L EQU *-OP_001 LET THE ASSEMBLER CALC LENGTH
* 
OP_002 DC C' REPORT DSNAME='
OP_002D EQU *-OP_002 LET ASSEMBLER CALCULATE LENGTH
DC CL44' '
DC C' DSORG='
OP_002C EQU *-OP_002
DC CL2' '
DC C' RECFM='
OP_002E EQU *-OP_002
DC CL2' '
DC C' LRECL='
OP_002F EQU *-OP_002
DC CL5' '
OP_002L EQU *-OP_002 LET THE ASSEMBLER CALC LENGTH

© 1997. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (940) 455 7050, fax (940) 455 2492.
OP003 DC C' MESSAGES DSNAME=' LET ASSEMBLER CALCULATE LENGTH
OP003D EQU '*-OP003 ALLOCATE SPACE FOR DSNAME
DC CL44' '
DC C' DSORG=' DATASET ORGANIZATION
OP003C EQU '*-OP003 RECORD TYPE
DC CL2' '
DC C' RECFM='
OP003E EQU '*-OP003 LOGICAL RECORD LENGTH
DC CL2' '
DC C' LRECL='
OP003F EQU '*-OP003 LET THE ASSEMBLER CALC LENGTH
DC CL5' '
OP003L EQU '*-OP003 *
OP004 DC C' SRCARGS DSNAME=' LET ASSEMBLER CALCULATE LENGTH
OP004D EQU '*-OP004 ALLOCATE SPACE FOR DSNAME
DC CL44' '
DC C' DSORG=' DATASET ORGANIZATION
OP004C EQU '*-OP004 RECORD TYPE
DC CL2' '
DC C' RECFM='
OP004E EQU '*-OP004 LOGICAL RECORD LENGTH
DC CL2' '
DC C' LRECL='
OP004F EQU '*-OP004 LET THE ASSEMBLER CALC LENGTH
DC CL5' '
OP004L EQU '*-OP004 *
OP005 DS XL1 SPACE FOR RECORD NUMBER
OP005R DS XL6 FILLER
OP005L EQU '*-OP005 LET THE ASSEMBLER CALC LENGTH
MAX5 EQU L'O_LINE-OP005L LET THE ASSEMBLER CALCULATE
OP006 DS XL1 SPACE FOR RECORD NUMBER
OP006R DS XL8 FILLER
DC C'RECDS READ FROM THE SYSUT1 DATASET'
OP006L EQU '*-OP006 LET THE ASSEMBLER CALC LENGTH
OP007 DS XL1 SPACE FOR RECORD NUMBER
OP007R DS XL6 FILLER
DC C'RECDS FOUND CONTAINING THE SEARCH ARGUMENTS'
OP007L EQU '*-OP007 LET THE ASSEMBLER CALC LENGTH
TITLE 'IEBIBALL - DCB RELATED INFORMATION'
UT1_DDN DC CL8'SYSUT1' USED BY THE DSABSERV ROUTINE
UT2_DDN DC CL8'REPORT' USED BY THE DSABSERV ROUTINE
UT3_DDN DC CL8'MESSAGES' USED BY THE DSABSERV ROUTINE
UT4_DDN DC CL8'SRCARGS' USED BY THE DSABSERV ROUTINE
* DECLARE THE DCB EXTENSIONS
DCBE_UT1 DCBE RMODE31-BUFF
DCBE_UT2 DCBE RMODE31-BUFF
DECLARE THE DCB INFO FOR THE FILES

UT1    DCB    DSORG=PS,MACRF=(GL),DDNAME=SYSUT1,EODAD=EOF_UT1, +
        SYNAD=SYN_UT1

UT2    DCB    DSORG=PS,MACRF=(PM),DDNAME=REPORT,DEVDA=DA,
         DCBE=DCBE_UT2,SYNAD=SYN_UT2

UT3    DCB    DSORG=PS,MACRF=(PM),DDNAME=MESSAGES,DEVDA=DA,
         DCBE=DCBE_UT3,SYNAD=SYN_UT3

UT4    DCB    DSORG=PS,MACRF=(GL),DDNAME=SRCHARGS,EODAD=EOF_UT4,
         DEVDA=DA,DCBE=DCBE_UT4,SYNAD=SYN_UT4

$ESASTG

@DSAB   DS   A   ADDRESS OF DSABSERV
RET_CODE DS   F   RETURN CODE FIELD
D_WORK   DS   D   WORK AREA
PLIST    DS   (4*5)A  USED BY $CALL
UT1_L    DS   H   LENGTH OF THE DSNAME
UT1_DSN   DS   XL44  SPACE FOR DATASET NAME
UT1_LREC   DS   XL2   SPACE FOR RECORD SIZE
UT1_DSO   DS   XL2   SPACE FOR DATASET ORG
UT1_RT    DS   XL1   SPACE FOR RECORD TYPE
UT2_L    DS   H   LENGTH OF THE DSNAME
UT2_DSN   DS   XL44  SPACE FOR DATASET NAME
UT2_LREC   DS   XL2   SPACE FOR RECORD SIZE
UT2_DSO   DS   XL2   SPACE FOR DATASET ORG
UT2_RT    DS   XL1   SPACE FOR RECORD TYPE
UT3_L    DS   H   LENGTH OF THE DSNAME
UT3_DSN   DS   XL44  SPACE FOR DATASET NAME
UT3_LREC   DS   XL2   SPACE FOR RECORD SIZE
UT3_DSO   DS   XL2   SPACE FOR DATASET ORG
UT3_RT    DS   XL1   SPACE FOR RECORD TYPE
UT4_L    DS   H   LENGTH OF THE DSNAME
UT4_DSN   DS   XL44  SPACE FOR DATASET NAME
UT4_LREC   DS   XL2   SPACE FOR RECORD SIZE
UT4_DSO   DS   XL2   SPACE FOR DATASET ORG
UT4_RT    DS   XL1   SPACE FOR RECORD TYPE
O_LINE    DS   XL133  OUTPUT LINE BUFFER
UT1_FLAG    DS   XL1   FLAG INDICATOR FOR DCB
UT2_FLAG   DS   XL1   FLAG INDICATOR FOR DCB
UT3_FLAG   DS   XL1   FLAG INDICATOR FOR DCB
UT4_FLAG   DS   XL1   FLAG INDICATOR FOR DCB
R_BXLE    DS   3A   USED BY THE BXLE SCAN LOOP
TRAN_TAB   DS  256XL1  USED BY THE TRT OPERATION
RECORD_R DS   PL6   NUMBER OF RECORDS READ
RECORD_M DS   PL4   NUMBER OF RECORDS FOUND
RECORD_N DS   PL4   CURRENT NUMBER
ARG_TB    DS   A   @(FIRST ARG IN THE TABLE)
ARG_TI    DS   A   TABLE INCREMENT
ARG_TE    DS   A   @(LAST ARG IN THE TABLE)
ARG_L     DS   H   LENGTH OF SEARCH ARG - 1
AND_FLAG  DS   XL1   FLAG FOR AND OPERATION
ARG_ARG DS XL80          SPACE FOR THE SEARCH ARG
ARG_ENTL EQU *-ARG_L     LET ASSEMBLER CALC LENGTH
ARG_NUM EQU 99           MAX NUMBER OF ARGUMENTS
       DS (ARG_NUM*ARG_ENTL)X11 ALLOCATE SPACE
ARG_TBLL EQU *-ARG_TB    CALCULATE TABLE SIZE
* PULL IN THE DCB MAPPING MACRO
DCBD DSORG=(QS)
END IEBIBALL

DSABSERV PROGRAM

TITLE 'DSABSERV - ACCESS DATASET JFCB INFORMATION'

* CSECT : DSABSERV
* MODULE : DSABSERV
* DESC : DSABSERV IS A CALLABLE ROUTINE THAT CAN BE USED TO OBTAIN
* THE NAME OF THE DATASET THAT IS ASSOCIATED WITH A DDNAME
* IN THE CURRENT STEP. RECORD TYPE, DATASET ORGANIZATION
* AND LOGICAL RECORD LENGTH ARE ALSO RETRIEVED. SOME OF
* FIELDS MAY NOT BE AVAILABLE IF THE DATASET HAS NOT BEEN
* OPENED. THE ROUTINE DOES NOT ESTABLISH A RECOVERY ENVIRON-
* MENT, SO IT WILL PERCOLATE IF IT ABENDS.
* MACROS : $ESAPRO $ESAEP $ESASTG GETDSAB SWAREQ
* DSECTS : IHADSAB CVT IEFJESCT IEFJTCB IEFJFCBN IEFZB505
* INPUT : NONE
* OUTPUT : NONE
* PLIST : R1 POINTS TO A STANDARD PARAMETER LIST
*   R1+X'00' ADDRESS OF HALFWORD FOR DATASET NAME LENGTH
*   R1+X'04' ADDRESS OF DDNAME
*   R1+X'08' ADDRESS OF 44 BYTE AREA TO PLACE THE DATASET
*       NAME INTO
*   R1+X'0C' ADDRESS OF A FULLWORD. FIRST HALFWORD CONTAINS
*       LRECL, SECOND HALFWORD CONTAINS DSORG
*   R1+X'10' ADDRESS OF 1 BYTE CONTAINING RECFM
*   THE PLIST IS VARIABLE IN LENGTH. THE HIGH ORDER BIT IS
*   TURNED ON IN THE LAST ADDRESS IN THE LIST. THIS ALLOWS
*   THE ROUTINE TO DETERMINE HOW MANY ARGUMENTS ARE IN THE
*   PLIST.

EJECT
DSABSERV $ESAPRO R12,RM=ANY,AM=31
   USING ZB505,R9       LET THE ASSEMBLER KNOW
   LR R8,R1             PICK UP POINTER FROM CALLER
   LTR R8,R8            Q. DID WE GET SOME PARMS
   BNZ GOT_PARM         A. YES, CALLER PASSED SOMETHING
   MVC RET_CODE,RC016   SET IN A RETURN CODE
   B EXITPROG           EXIT THE ROUTINE

* BUILD THE TRANSLATE TABLE. IT IS USED TO DETERMINE THE LENGTH OF
* THE DATASET NAME. ONLY SIGNIFICANT CHARACTER IS THE SPACE X'40'.

GOT_PARM DS 0H
MVI TRANTAB+C' ','C'
PUT SPACE IN Xlate TABLE

NXT_PARM DS 0H
LM R3,R7,0(R8)
PICK UP ADDRESSES FROM CALLER
* R3 NOW HAS @(DSNAME LENGTH)
* R4 NOW HAS @(DDNAME)
* R5 NOW HAS @(DSNAME)
* R6 NOW HAS @(RECORD LENGTH,
   DS ORGANIZATION)
* R7 NOW HAS @(RECORD TYPE)

XC EPA_AREA,EPA_AREA
INSURE AREA IS CLEARED
LA R9,EPA_AREA
GET @(EPA AREA)

* SET THE DSNAME LENGTH TO THE MAXIMUM POSSIBLE, AND PRIME THE DSNAME *
* FIELD WITH ASTERISKS. IT WILL BE UP TO THE CALLER TO CHECK THE *
* CONTENTS OF THE DSNAME FIELD TO SEE WHAT IT CONTAINS.

MVC 0(Z,R3),HALF44
SET MAX DSNAME LENGTH
MVI 0(R5),C'*'
DUMMY OUT FIRST BYTE OF THE
* DATASET NAME FIELD
MVC 1(R4,R5),0(R5)
DUMMY OUT THE REMAINDER OF
* THE DATASET NAME FIELD

* UTILIZE THE GETDSAB SERVICE TO GET THE ADDRESS OF THE DATA SET *
* ASSOCIATION BLOCK. FROM THE DSAB, WE PICK UP THE POINTER TO THE *
* TIOT ENTRY. FROM THE TIOT ENTRY, WE PICK UP THE SVA FOR THE JFCB. *
* THEN WE USE THE SWAREQ SERVICE TO GET THE ADDRESS OF THE JFCB, AND *
* FROM THERE WE PICK UP THE DATASET NAME.

GETDSAB DDNAME-(R4),DSABPTR-PTRDSAB,RETCODE-DSAB_RET,
RSNCODE-DSAB_RSN,MF-(E.DYN_DSAB)
CLC DSAB_RET,RC000
Q. DO WE HAVE THE DSAB
BNE NXT_NTRY
A. ENCOUNTERED AN ERROR
L R4,PTRDSAB
GET @(DSAB)
L R4,DSABTIOT-DSAB(.R4)
GET @(TIOT ENTRY)

* FROM THE TIOT ENTRY FOR THE DDNAME IN QUESTION WE PICK UP A TOKEN *
* THAT WILL BE PLACED INTO THE EPA (EXTENDED PARAMETER AREA) THAT WILL*
* BE PASSED TO SWAREQ.

MVC SWRAL*(L'TIOEJFCB),TIOEJFCB-TIOENTRY(R4)
LA R4,EPA_AREA
GET @(EXTENDED PARAMETER AREA)
ST R4,SVA_PTR
SET UP PLIST FOR CALL TO SWAREQ
SWAREQ FCODE-RL,EPA-SVA_PTR,UNAUTH-YES,MF-(E.DYN_SWA)
C R15,RC000
Q. CLEAN FROM SWAREQ
BNE NXT_NTRY
A. ENCOUNTERED AN ERROR
L R1,SWBLKPTR
GET @(JFCB)
MVC 0(Z,R6),JFCLRECL-JFCBDSTC(R1)
GET THE RECORD LENGTH
MVC 2(Z,R6),DST_##
PRIME WITH UNKNOWN
TM JFCDSRG1-JFCBDSTC(R1),JFCORGPS Q. PHYSICAL SEQUENTIAL
BNO CHK_PO A. NO, GO SE IF PO
MVC 2(2,R6),DST_PS INDICATE PS FILE TYPE
B CHKRECFM GO DETERMINE RECORD TYPE

CHK_PO DS ØH
TM JFCDSRG1-JFCBDSCT(R1),JFCORGPO Q. PARTITIONED ORG.
BNO CHKRECFM A. NO, ?? FILE TYPE
MVC 2(2,R6),DST_PO INDICATE PO FILE TYPE

CHKRECFM DS ØH
MVC Ø(1,R7),U_TYPE# SET TO UNDEFINED
TM JFCRECFM-JFCBDSCT(R1),JFCUND Q. UNDEFINED
BNO CHK_FIX A. NO
MVC Ø(1,R7),U_TYPE SET TO UNDEFINED
B MVC_DSN GO MOVE DSN

CHK_FIX DS ØH
TM JFCRECFM-JFCBDSCT(R1),JFCFIX Q. FIXED RECORD TYPE
BNO CHK_VAR A. NO
MVC Ø(1,R7),F_TYPE SET TO FIXED
B MVC_DSN GO MOVE DSN

CHK_VAR DS ØH
TM JFCRECFM-JFCBDSCT(R1),JFCVAR Q. VARIABLE
BNO MVC_DSN GO MOVE DSN
MVC Ø(1,R7),V_TYPE SET TO VARIABLE

MVC_DSN DS ØH
MVC Ø(L'JFCBDSMN,R5),JFCBDSMN-JFCBDSCT(R1) MOVE THE DSNAME
TO THE CALLER'S AREA

* TRT Ø(L'JFCBDSMN,R5),TRANTAB SCAN FOR THE FIRST BLANK
* IN THE DATASET NAME
BC 2,NXT_NTRY NO BLANKS ENCOUNTERED
BC 4,CALC_LEN BLANK FOUND, CALCULATE LENGTH
* SHOULD NEVER FALL THROUGH, BUT
* JUST IN CASE WE DO
MVC RET_CODE,RC004 SET A RETURN CODE TO INDICATE
B EXITPROG LEAVE THE ROUTINE
* AN ERROR WAS ENCOUNTERED

CALC_LEN DS ØH
SR R1,R5 CALCULATE DSNAME LENGTH - 1
SIL R1,Ø(R3) PUT IT IN CALLERS STORAGE
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* CONTINUE UNTIL WE HAVE PROCESSED THE LAST TRIPLET OF ADDRESSES. *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

NXT_NTRY DS ØH
TM HI_BITL(R8),HIBITON Q. LAST SET OF ARGUMENTS
BO EXITPROG A. YES, ALL DONE
LA R8,PTR_ADJ(R8) ADJUST REGISTER 2
B NXT_PARM GO PROCESS NEXT SET

EXITPROG DS ØH
$ESAEPI RET_CODE GET THE RETURN CODE

PTR_SIZE EQU 4 SIZE OF 1 PARAMETER
PTR_NUM EQU 5 NUMBER OF PARMS/ARGUMENT
PTR_ADJ EQU PTR_SIZE*PTR_NUM INCREMENT SIZE
HI_BITL EQU PTR_ADJ-4 LOCATION TO CHECK FOR HIGH BIT

HIBITON EQU X'80'  
RC000 DC F'0'  
RC004 DC F'4'  
RC016 DC F'16'  
HALF44 DC H'44'  
DST_PS DC CL2'PS'  
DST_PO DC CL2'PO'  
DST_## DC CL2'??'  
F_TYPE DC CL1'F'  
V_TYPE DC CL1'V'  
U_TYPE DC CL1'U'  
U_TYPE# DC CL1'?'  

USED FOR ADDRESS TESTING
USED FOR RETURN CODE SETTING
USED FOR RETURN CODE SETTING
USED FOR RETURN CODE SETTING
MAX DATASET NAME LENGTH
PHYSICAL SEQUENTIAL FILE
PARTITIONED ORGANIZATION
DON'T KNOW THE FILE TYPE
FIXED RECORD TYPE
VARIABLE RECORD TYPE
UNDEFINED RECORD TYPE
UNKNOWN RECORD TYPE

RETURN CODE FROM GETDSAB
REASON CODE FROM GETDSAB
USED BY THE GETDSAB CALL
RETURN CODE FIELD
POINTER TO THE EPA
SPACE FOR THE SWAREQ EPA
SET ASIDE SPACE FOR THE

* TRANSLATE TABLE

SET ASIDE SPACE FOR THE GETDSAB MACRO
GETDSAB MF-(L,DYN_DSAB)

SET ASIDE SPACE FOR THE SWAREQ MACRO
DYN_SWA SWAREQ MF-L

TITLE 'DSABSERV - MAP OUT THE DSAB CONTROL BLOCK'
IHADSAB
TITLE 'DSABSERV - MAP OUT THE CVT CONTROL BLOCK'
CVT DSECT-YES,LIST-YES
TITLE 'DSABSERV - MAP OUT THE JESCT CONTROL BLOCK'
IEFJESCT
TITLE 'DSABSERV - MAP OUT IEFZB5O5'
IEFZB5O5
TITLE 'DSABSERV - MAP OUT THE TIOT CONTROL BLOCK'
TIOT DSECT
IEFIOT1
TITLE 'DSABSERV - MAP OUT THE JFCB CONTROL BLOCK'
JFCBSDCT DSECT
IEFJFCBN

END DSABSERV  
TELL ASM WHERE PROGRAM ENDS

$ESAPRO MACRO

MACRO
&LABEL $ESAPRO &AM=31,&RM=ANY,&MODE=P

-----------------------------------------------

THIS MACRO WILL PROVIDE ENTRY LINKAGE AND OPTIONALLY
MULTIPLE BASE REGISTERS. TO USE THIS MACRO, YOU NEED TO
ALSO USE THE $ESASTG MACRO. THE $ESASTG DEFINES THE SYMBOL
QLENGTH WHICH OCCURS IN THE CODE THAT $ESAPRO GENERATES.

IF YOU DO NOT CODE ANY OPERANDS, THEN REGISTER 12 WILL BE
USED AS THE BASE. IF YOU CODE MULTIPLE SYMBOLS, THEN THEY
WILL BE USED AS THE BASE REGISTERS.

EXAMPLES:

SECTNAME $ESAPRO 5 - REG 12 BASE
SECTNAME $ESAPRO 5 - REG 5 BASE
SECTNAME $ESAPRO R10, R11 - REGS 10 AND 11 ARE BASES

**********************************************************************

LCLA &AA, &AB, &AC

R0 EQU 0
R1 EQU 1
R2 EQU 2
R3 EQU 3
R4 EQU 4
R5 EQU 5
R6 EQU 6
R7 EQU 7
R8 EQU 8
R9 EQU 9
R10 EQU 10
R11 EQU 10
R12 EQU 12
R13 EQU 13
R14 EQU 14
R15 EQU 15
RF EQU 15

FPR0 EQU 0
FPR2 EQU 2
FPR4 EQU 4
FPR6 EQU 6

&LABEL CSECT
&LABEL AMODE &AM
&LABEL RMODE &RM

SYSSTATE ASCENV-AMODE

SET THE ENVIRONMENT

B $$$$$EYEC-*(R15) BRANCH AROUND EYECATCHER
DC ALI($$$$$EYEC-*)-1) EYECATCHER LENGTH
DC CL8'&LABEL' MODULE ID
DC CL3' - '
DC CLB'&SYSDATE' ASSEMBLY DATE
DC CL3' - '
DC CLB'&SYSTIME' ASSEMBLY TIME
DC CL3' ' FILLER

$$$$F1SA DC CL4'FISA' USED FOR STACK OPERATIONS
$$$$4096 DC F'4096' USED TO ADJUST BASE REGS

$$$$EYEC DS 0H

BAKR R14,0 SAVE GPRS AND ARS ON THE STACK
AIF (N'&SYSLIST EQ 0).USER12
LAЕ &SYSLIST(1),0(R15,0) LOAD OUR BASE REG
USING &LABEL,&SYSLIST(1) LET THE ASSEMBLER KNOW
AGO .GNBASE

.USER12 ANOP
MNOTE *, 'NO BASE REG SPECIFIED, REGISTER 12 USED'
LAЕ R12,0(R15,0) LOAD OUR BASE REG
USING &LABEL,R12 LET THE ASSEMBLER KNOW
AGO .STGOB

.GNBASE ANOP
AIF (N'&SYSLIST LE 1).STGOB
&AA SETA 2
&AC SETA 4096

&GNBASE1 ANOP

&AB SETA &AA-1
LR &SYSLIST(&AA),&SYSLIST(&AB) GET INITIAL BASE
A &SYSLIST(&AA),$$$$4096 ADJUST NEXT BASE
USING &LABEL+&AC,&SYSLIST(&AA) LET THE ASSEMBLER KNOW

&AA SETA &AA+1
&AC SETA &AC+4096
AGO .GNBASE1

&STGOB ANOP

L R0,QLENGTH GET THE DSECT LENGTH

STORAGE OBTAIN,LENGTH-(R0),LOC-(RES,ANY)

LR R15,R1 GET @(OBTAINED AREA)
L R13,QDSECT GET DISPLACEMENT INTO AREA
LA R13,0(R13,R15) GET @(OBTAINED AREA)
LR R0,R13 SET REG 0 - REG 13
L R1,QLENGTH GET THE LENGTH OF THE AREA
XR R15,R15 CLEAR REG 5
MVCL R0,R14 INITIALIZE THE AREA
MVC 4(4,R13),$$$$F1SA INDICATE STACK USAGE
USING DSECT,R13 INFORM ASSEMBLER OF BASE

.MEND ANOP
$ESAEPI MACRO

MACRO
$ESAEPI

******************************************************************************
** THIS MACRO WILL PROVIDE EXIT LINKAGE. IT WILL FREE THE
** STORAGE AREA THAT WAS ACQUIRED BY THE $ESAPRO MACRO. YOU
** CAN Optionally PASS IT A RETURN CODE VALUE. THIS VALUE IS
** EITHER THE LABEL OF A FULL WORD IN STORAGE, OR IT IS A REG-
** ISTER. AS WITH THE $ESAPRO MACRO, YOU NEED TO USE THE $ESASTG
** MACRO. THE SYMBOL QLENGTH WHICH OCCURS IN THE CODE THAT IS
** GENERATED BY THIS MACRO IS DEFINED BY $ESASTG
**
** EXAMPLES:
**   $ESAEPI - NO RETURN CODE SPECIFIED
**   $ESAEPI (R5) - RETURN CODE IS IN REG 5
**   $ESAEPI RETCODE - RETURN CODE IS IN THE FULLWORD AT
**                     RETCODE
******************************************************************************

AIF (N'&SYSLIST EQ 0).STGFRE
AIF ('&SYSLIST(1)'(1,1) EQ '').REGRC
L R2,&SYSLIST(1) GET RETURN CODE VALUE
AGO .STGFRE
.REGRC ANOP
LR R2,&SYSLIST(1,1) GET RETURN CODE VALUE
.STGFRE ANOP
L R0,QLENGTH GET THE DSECT LENGTH
STORAGE RELEASE,LENGTH=(R0),ADDR=(R13)
AIF (N'&SYSLIST NE 0).SETRC
XR R15.R15 CLEAR THE RETURN CODE
AGO .MEND
.SETRC ANOP
LR R15,R2 SET THE RETURN CODE
.MEND ANOP
PR RETURN TO CALLER

* FOR ADDRESSABILITY PURPOSES
LTORG
MEND

$ESASTG MACRO

MACRO
$ESASTG

******************************************************************************
** THIS MACRO IS USED IN CONJUNCTION WITH THE $ESAEPI AND $ESAPRO
** MACROS. IT PROVIDES A Q TYPE ADDRESS CONSTANT WHICH WILL CON-
THE LENGTH OF THE DSECT. A REGISTER SAVE AREA ID PROVIDED AS
WELL.

EXAMPLES:
$ESASTG
XXX DC F  — DEFINE ADDITIONAL STORAGE AREA
YYY DC XL255

**********************************************************************

ODSECT DC Q(DSECT)  DEFINE A QCON
OLENGTH CXD  LET ASM CALCULATE THE LENGTH
DSECT DSECT
DS 18F  SET ASIDE REGISTER SAVE AREA

$CALL MACRO

MEND
MACRO
&NAME $CALL &ENTRY,&OPRNDS,&VLPARA,&BM-BALR,&ID=,&MF=
**********************************************************************

.* MODIFIED VERSION OF THE IBM SUPPLIED CALL MACRO  *
**********************************************************************

GBLB &IHBSWA,&IHBSWB
GBLC &IHBN0
LCLC &GNAME
&IHBN0 SETC '309'
&GNAME SETC 'IHB'.('&SYSNDX'
&IHBSWA SETB ('&VLPARA' EQ 'VL')
&IHBSWB SETB ('&ENTRY' EQ '(15)')
AIF ('&VLPARA' NE '' AND '&VLPARA' NE 'VL').ERROR4
AIF ('&MF' EQ 'L' AND '&ENTRY' NE '').ERROR1
AIF ('&MF' EQ 'L' AND '&ID' NE '').ERROR2
AIF ('&MF' NE 'L' AND '&ENTRY' EQ '').ERROR3
&NAME DS 0H  ALIGNMENT
AIF ('&MF' EQ 'L').CONTC
AIF (&IHBSWB).CONTCC
.CONTC AIF ('&OPRNDS' EQ '' AND
('&MF' EQ 'I' OR '&MF' EQ 'L')).CONTB
.CONTA IHBOPLTX &ENTRY,&OPRNDS,&NAME,MF-&MF
.CONTB AIF ('&MF' EQ 'L').EXIT
AIF (&IHBSWB).CONTD
L 15,&ENTRY  LOAD 15 WITH ENTRY ADR
.CONTD AIF ('&BM' EQ 'BASSM').CONTE
BALR 14,15  BRANCH TO ENTRY POINT
AGO .CONTF
.CONTE BASSM 14,15  BRANCH TO ENTRY POINT
.CONTF AIF ('&ID' EQ '').EXIT
DC X'4700'  NOP INSTRUCTION WITH
Since MVS Version 5, an MVS command exit has been made available as a standard exit point. By that time many sites had home-grown versions of programs that would listen in on the subsystem interface, intercept commands, and respond to MVS. This was a somewhat complex piece of code to write, and all of this has been made much easier by making use of the published exit point. The exit has to be in a LNLKSTed dataset. It also has to be re-entrant and receives control in supervisor state key 0. It can have any name complying with standard load module naming conventions and is defined to MVS via the MPFLSTxx member in the following way:

```
.CMD USEREXIT(exitname)
```

It is dynamically refreshable by relinking the module into the LNKLST-library followed by:

```
T MPF=xx
```

where xx the suffix of the MPFLSTxx member in SYS1.PARMLIB (or any other SYSx.PARMLIB as from OS/390). This way it is really easy to add changes to the exit and no pre-loading in common storage or zapping of pointers in memory is required. The module is also ESTAE-protected and a catastrophic error in the module will merely
disable the exit. (Keep in mind that the exit is called in supervisor state
0 though, so it is quite easy to do irrecoverable damage to the operating
system if care is not taken.)

The exit can be used to alter the command. If a command is altered,
both the old command and the new command are displayed on the
console (and on the SYSLOG), but only the altered command is
executed. We will look at a few uses of this facility and also at some
coding hints.

When the module is called, a copy of the command amongst other
things is passed to the routine. This is done for all commands, so a
command to any of the other subsystems can be viewed, altered, or
denied even if it has a prefix character assigned to it. An important
thing to remember is that this command exit could potentially lock
itself in. That is, if coded incorrectly, the command required to disable
it (TMPF=NO) can also be rejected – making an IPL the only way to
recover from an infinite loop in the module. It is good practice to scan
the text for any TMPF commands right at the start of the logic and,
if found, to immediately return to MVS with a return code of 0. This
way we can be sure that the TMPF command is always processed.

Another good idea is to make the exit merely a text analyser with all
the actual work being done in called subroutines. When we receive a
copy of the command BUFFER, look for our command(s) by comparing
them to a table where we keep all the ones we are interested in. If we
find a match, we set up our own ESTAE and then do a LINK
EP=module for the particular function. This way we end up with
several independent load modules, leading to a clean modular design.
By doing this we can develop new command modules and, if they
abend (as modules tend to do whilst being developed or tested), we
intercept the abend and recover. We then never get our exit disabled
by MVS because the exit itself never abends, only one of its subroutines
for which we have set up an ESTAE. We can make use of a bit pattern
or a flag in our command table to indicate that a certain command is
causing an ABEND, and from this we can issue a warning message
should the command be entered again. The following is a suggested
sequence of events in the main routine:

1. Set up addressability to the passed command text (see example
later).
Because this module has to be re-entrant, obtain storage in subpool 229 for its workareas.

Remove all blanks from the command buffer to standardize the format.

See if the command buffer contains the text we are looking for by comparing it with our table of commands.

If it does not, return to MVS with a return code of 0 (telling MVS to proceed).

If it does, do the following:

- Set up an ESTAE environment.
- Call the matching command processing subroutine for that particular command.
- Decide if MVS should further process the command or not.
- Return to MVS with a 0 (proceed) or 4 (ignore). Ignore would be the case if our logic has already done the necessary work or if we decide to reject the command for some reason.

Keep in mind that the command exit also gets a copy of all messages sent through the system. An infinite loop could potentially be created should we issue a message containing the text we are scanning for in the command buffer.

We will now look at a few uses of this command exit and then work through the above four points with examples and some tips. The following are ideas of what we may want to do in a command exit:

- Refreshing a single LLA – dataset is cumbersome (we have to update a PARMLIB member or have one ready for it), so most systems programmers simply enter F LLA,REFRESH. This places a massive overhead on the system and in some cases can lead to performance problems for quite some time because VLF is also involved in the process. A much better idea would be to have the ability to enter a command of the format:

  F LLA,REFRESH~mydsname

  Because we have the LLACOPY macro available, this is quite a
simple process once we have identified the dataset name from the command text. As our routine will be doing the LLACOPY work itself, we can return to MVS with a return code of four which will cause MVS to not process the command any further – that is, LLA never gets instructed by MVS to actually do the refresh. (One of the drawbacks of this exit is that people become used to it: if we now get it disabled for some reason, MVS will pass the above command for further processing which of course does not fit in with the standard format. This is the reason why you should make sure that once in use, the exit itself never gets disabled through an abend.)

- Inspecting and possibly restricting VARY commands. With the introduction of 4-digit commands, an incorrectly entered VARY command can cause quite some overhead on a system. The command

```
V 123-456,ONLINE
```

incorrectly entered as

```
V 123-4566,ONLINE
```

(due to a typo) will hang MVS for quite some time. It may be a good idea to investigate command ranges and only pass them through to MVS (by means of a 0 return code) if they fall within reasonable ranges.

- Inspecting the:

```
E jobname,SRVCLASS~name
```

may be a good idea. It is also a good idea to have a RACF-routine for any of the new commands introduced. This same routine can be used to verify access to certain restricted commands. First do a RACF-check and only allow the command to be issued if the user is within a certain group or has certain RACF privileges.

- Any product that manipulates UCBs to facilitate tape sharing could potentially leave the UCBs in an incorrect state if it abends or is FORCEd out of the system – requiring a zap in the UCB, which is a dangerous practice even at the best of times. A new UCBZAP command can be introduced with a module doing the
work for us. (This one would definitely require the RACF–check first because it could be extremely destructive.) Any other high-risk zaps that systems programmers have to do from time to time could be put into the command exit. It is far better to code up the exit accurately and with a cool head than to have to work out offsets and set up a zap during a time when the system is experiencing an emergency of sorts.

• RMF has a routine that can be called to obtain figures on service consumption, real and auxiliary frame usage, etc. This module is called ERBSMFI. Using the command exit we can define a new command, something like D BUSY, which can then have a module called in which we invoke ERBSMFI and manipulate its output. In a case of a total hang (no TSO user or monitor gets dispatched), we may be able to find the cause by entering a D BUSY command from the console. The routine should be written in such a way that it look for high consumers of CPU etc. (The way to process ERBSMFI is to call it, save the returned values, wait a few milliseconds, call it again and then make decisions based on the differences obtained. For instance, if address space ABC had used X CPU seconds at the time of the first call and Y CPU seconds at the time of the second call, then Y - X will show us how many CPU seconds it has used. One has to take the number of processors on–line into consideration to be able to express this as a CPU % – the SDSF source code is a good example to look at.)

• GRS contention is very common in the early stages of sysplex implementation. The D GRS,C command (and other versions of it) goes some way to help resolve contention. There is however a fair bit more information available by doing a GQSCAN macro. This will for instance show which member of a sysplex has a RESERVE on a volume. By scanning through this information and looking at I/O queues chained off UCBs, one can greatly enhance the systems programmer’s ability to resolve problems during sysplex hang situations. So it may be a good idea to have something like D RESERVE to show which sysplex member is causing the problem.

• In the October 1997 issue of MVS Update an example was given
on how to write a routine to display disk characteristics. This routine could easily be adopted to support a command such as DISKTYPE xxxx, displaying more information regarding a disk unit directly onto the console.

There are more good reasons to have a command exit installed, but by now you should have an idea of the benefits that can be derived from it. It also gives a large degree of flexibility once it is in place – if a certain command suddenly has to be intercepted for one reason or another it could be a fairly simple task to make an addition to the exit, provided it was well planned and structured as suggested.

We will now get back to points 1 – 4 mentioned earlier and give some examples of how this can be implemented.

1 To set up addressability to the passed buffer, the following can be used. When we receive control, register 1 contains the address of the command installation exit routine parameter list mapped by the macro IEZVX101. A large amount of information is contained in this DSECT and it includes fields such as:

- CMDXISYN – the name of the system that issued the command.
- CMDXCNNM – the name of the console that issued the command.
- CMDXTOKN – command issuer TOKEN.
- CMDXCLIP – pointer to the command length and the command image.

(By making use of the SHOWMEM routine published in the May 1997 issue of MVS Update, it may be a good idea to display some of these fields and also the command buffer before making decisions based on their contents.)

Sample text to get to the command buffer:

```
L R4,0(R1) .Passed pointer when we receive control
USING CMDX,R4 .Addressability to passed parameter
L R4,CMDXCLIP .Command buffer address
DROP R4
USING CMDXCLIB,R4 .Addressability to the command buffer
LH R5,CMDXCMDL .Length of entered command
```
As mentioned before, it may be a good idea to de-blank the command buffer before we start. Keep in mind that we can alter a command by overlaying the command buffer and setting a flag (the field name is CMDXRFLI and the flag is CMDXRCMI) so it is best to copy the command buffer into our own workarea before we start manipulating it. We now remove all the blanks by going through a simple loop (make sure you do not exceed the length of the passed command because this will lead to an 0C4, which will disable the exit). Once we have de-blanked the command, we can enter another loop, comparing it to a table with our customized commands. If we decide to alter the command we can then move it back into the original command buffer that was passed to us.

Some commands will never be passed to MVS, some commands will always be passed to MVS once we have taken note of them or altered them, and some may be passed to MVS if we are satisfied with the syntax (eg the range of a VARY command). By passing a return code of 0 to MVS the command gets processed and a return code of 4 instructs MVS to ignore the command (without giving any error message). Make sure that the successful processing of a private command resulting in a return code 0 does not cause the return code to be passed back to MVS because this will mean that MVS will then also try to interpret it. It may be a good idea to keep the return code that should be passed back to MVS in the command table. AX’00’ could mean that the command is always passed on, a X’04’ that it will never be passed on, and a X’02’ that the program logic will decide whether or not the command will ever reach MVS. Here is a sample of what a command table could look like:

```
CommTble DS OF .Command table
Com0001 DC C'FLLA,REFRESH=' .Deblanked format of command
Leng0001 DC AL2(*-Com0001) .Length of command text
EnPt0001 DC AL4(LLAEntpt) .Address of routine to call
RC0001 DC H'4' .Never pass command to MVS
*
Com0002 DC C'V' .Vary command
Leng0002 DC AL2(*-Com0002) .Length of command text
EnPt0002 DC AL4(VARYENTP) .Address of routine to call
RC0002 DC H'2' .May pass to MVS
```
Coding an ESTAE routine is a little complex. Keep in mind that we should actually return to MVS at the end of the routine and not to a point inside our program. The sequence of flow in the case of an abend is this: after the abend MVS gets control, it then branches off to our ESTAE routine which can do a clean-up, set a flag (e.g., mark the command as not available in a bit map), and/or write a message. Our ESTAE routine then returns back to MVS, telling it by means of the SETRP macro to either percolate (abend further, which in our case will have the entire command exit disabled) or branch back to a point in our mainline code. To be able to address our own storage area in the ESTAE routine we have to set up what is known as a RUBLIST. This list instructs MVS which of our registers to reload before giving control to the ESTAE routine. The best convention to ensure that we correctly return control to MVS from inside the ESTAE routine is to make use of the BAKR/PR instructions at the start and end of the routine.

Many automation packages offer high-level language interfaces to commands and messages generated and it is not suggested that the command exit is introduced to replace any of these. It has as a drawback that it somewhat exposes the system to any programming errors it may have. Once stabilized, it is however a handy and very powerful tool in the hands of a careful systems programmer. It also puts the control back where it belongs – with the MVS systems programmer (although the merits of this may be disputed by some). The command exit gets to look at incoming commands first and is in a position to override it or alter the syntax before it is seen by any of the other subsystems.

A A Keyser
Systems Programmer
Houghton Consulting Services Pty Ltd (Australia) © Xephon 1997
Year 2000 aid: list YEAR2K qualifying records

This program, YEAR2KLM, reads the selection file (OUTPUT) from program YEAR2K (see MVS Update issue 134), reformats it so that the source record is contiguous, and lists the records. This listing is useful in the following two ways:

- as a guide for the manager or lead analyst to determine quickly whether the qualified records need to be addressed, and, if so, the priority and resources that should be assigned.
- as a source for such assignments.

To address these different functions, a single option may be specified. This option is used to determine if the records for each member is to be listed on separate pages. This option is used when distributing information to individuals for conducting further study or as maintenance assignments. This option is selected by specifying the following PARM= statement:

```
PARM= 'SEPARATE'
```

It is recommended that both of these options be used with at least one of the copies being used for the initial analysis and for notes on tracking progress and the other forms for distributing to individual maintenance analysts for necessary changes. The original file may also be edited and notes of assignment etc be made prior to such listings. In this later case, it is recommended that such notes be restricted to the first 72 bytes of the record, since the remainder of the record is formatted based on positions 73-80 being non-blank (ie containing a member name). A sample of a listing, showing manual notes, is given in Figure 1.

SAMPLE JCL

```bash
!/SYST0021 JOB ...
!/-------------------------------------------------------------*//
!/STEP1 EXEC PGM=YEAR2KLM
!/SYSABEND DD SYSOUT=*  
!/SYSPRINT DD SYSOUT=*  
!/PRINTER DD SYSOUT=*  
!/INPUT DD DSN=SYST002.YEAR2K.MATCHES,DISP=SHR
!/```
Listing of Year2K Selections  Job=Syst0021  Dsn=Syst002.Year2K.Matches  Page 1

Member Record
1...5...10...15...20...25...30...35...40...45...50...55...60...65...70...75...80

********** NOTE: THE WORD ACRONYM IS A FALSE SELECTION BECAUSE ITS SUFFIX IS 'YM'
********** NOTE:
********** NOTE: Assigned to Prog001 for review and correction. KHN 11/18/96.
********** NOTE:

Aagi0010 73 007030 ACRONYM, (AC or KP at time of writing) depending
Aagi0010 171 0070170 02 Slashed-Year Pic 9(2).
Aagi0010 176 0070176 01 Workdate-Yy Mmdd.
Aagi0010 177 0070177 02 Workdate-Yy Pic X(2).
Aagi0010 181 0070181 01 Workdate-Mm/dd/Yy.
Aagi0010 186 0070186 05 WorkDatesl-Yy Pic 99.
Aagi0010 206 002060* "Julgreg" or "Gregjul" routines (conversion of Julian
Aagi0010 207 002070* dates to Gregorian, and vice versa).
Aagi0010 209 002090 01 Julian-Parm Pic X(23).
Aagi0010 210 002091 01 Filler redefines Julian-Parm.
Aagi0010 211 002100 05 Julian-Parm-Packed Pic 9(5) Comp-3.
Aagi0010 212 002102 05 Julian-Parm-Yy Mmdd Pic X(6).
Aagi0010 213 002110 05 Julian-Parm-Mm/dd YY Pic X(6).
Aagi0010 214 002140 05 Julian-Parm-Mm/dd/LY Pic X(8).
Aagi0010 224 002240* Compute-Date-and-Time routine.
Aagi0010 228 002280 01 Julian-Cvrt-Date Pic 9(7).
Aagi0010 229 002281 01 Filler redefines Julian-Cvrt-Date.
Aagi0010 231 002310 05 Currdte-Julian Pic 9(5).
Aagi0010 232 002320 05 Filler redefines Currdte-Julian.
Aagi0010 233 002330 10 Currdte-Julian-Yy Pic 9(2).
Aagi0010 234 002340 10 Currdte-Julian-Ddd Pic 9(3).
Aagi0010 237 002370 01 Currdte-Mm/ddyy.
Aagi0010 240 002400 05 Currdte-Yy Pic 9(2).
Aagi0010 242 002420 01 Currdte-Sl-Mm/ddyy.
Aagi0010 247 002470 05 Currdte-Sl-Yy Pic 9(2).
Aagi0010 249 002490 01 Currdte-Yy Mmdd Pic X(6).
Aagi0010 337 003370 02 L2 Pic X(75) Value 'at this time of the year.'
Aagi0010 982 009820 Move Currdte-Yy Mm/ddyy to Glje-Btbd-Batch-Entry-Date.
Aagi0010 1227 012270* Convert Julian date to Calendar date
Aagi0010 1228 012280 01 Move spaces to Julian-Parm.

1...5...10...15...20...25...30...35...40...45...50...55...60...65...70...

********** NOTE: THE WORD ACRONYM IS A FALSE SELECTION BECAUSE ITS SUFFIX IS 'YM'
********** NOTE:
********** NOTE: NO CORRECTION NECESSARY.

Member Record
1...5...10...15...20...25...30...35...40...45...50...55...60...65...70...

Figure 1: Year2KLM Sample report page
PROGRAM SOURCE

LCLC &MYNAME

* &MYNAME SETC 'YEAR2KLM' CSECT NAME
RBASE EQU 12 BASE REGISTER FOR CSECT
RBAL EQU 10 BAL REGISTER

* TITLE '&MYNAME' LISTING TITLE
***********************************************************************
*** THIS PROGRAM LISTS THE RECORDS SELECTED BY THE YEAR 2000 ***
*** ANALYSIS PROGRAM (YEAR2K). ***
***********************************************************************
EJECT
***********************************************************************
*** LINKAGE CONVENTIONS ENTERING PROGRAM ***
***********************************************************************

&MYNAME CSECT
NAME BEGIN STM R14,R12,12(R13) SAVE REGS TO CALLER S.A.
B (BEGIN-&MYNAME)(R15) BRANCH AROUND EYECATCHER
DC A('NAME') LENGTH OF CSECT NAME
NAME DC 'MYNAME' CSECT NAME
DC C'&SYSDATE &SYSTIME ' ASSEMBLY DATE/TIME STAMP
BEGIN LR RBASE,R15 LOAD BASE REGISTER
USING &MYNAME,RBASE ADDRESSABILITY
PRINT NOGEN
GETMAIN R,LV=WORKDLEN GET SAVE/WORK AREA
ST R1,8C(R13) MY S.A. ADDR INTO CALLER S.A.
ST R13,4(0,R1) CALLER S.A. ADDR INTO MY S.A.
LR R13,R1 R13 POINTS TO MY S.A.
USING WORKD,R13 ADDRESSABILITY OF SAVE AREA
L R1,4(0,R13) R1 POINTS TO CALLER S.A.
LM R15,R1,16(R1) R15 R0 AND R1 ARE RESTORED
EJECT

***********************************************************************
*** MAINLINE ROUTINE ***
***********************************************************************

MAIN EQU * BEGIN MAINLINE ROUTINE
ST R1,R1SAVE SAVE INITIAL R1
MVC PARM,=8C' ' SET TO PARAMETER AREA TO BLANKS
L R1,0(R1) LOAD ADDRESS OF PARAMETER
LH R8,0(R1) SET LENGTH
BCTR R8,0 DECREMENT TO LENGTH - 1
LTR R8,R8 WAS PARAMETER PRESENT?
BM MAINNOP NO
CH R8,=H'7' PARAMETER TOO LONG?
BH MAINNOP YES
EX R8,MOVEPARM MOVE PARAMETER TO SAVE AREA

* MAINNOP XC COMPCODE,COMPCODE CLEAR COMPLETION CODE
MVC JGMOTBL(13*JGMOTBL),JGMOTBLD COPY JUGREG DAYS/MONTH

* BEGIN DCB INITIALIZATION

MVC PRINTERN(PRINTERL),PRINTERD INITIALIZE DCB
MVC INPUTN(INPUTL),INPUTD INITIALIZE INPUT DCB

* END DCB INITIALIZATION
* BEGIN DCB OPENS
MVC PROPENL(PROPENLN),OPEND INITIALIZE SET PRINTER OPEN LIST
OPEN (PRINTER,(OUTPUT)),MF=(E,PROPENL) OPEN PRINTER
MVC IPOPENL(IPOPENLN),OPEND SET INPUT OPEN LIST
OPEN (INPUT,(INPUT)),MF=(E,IPOPENL) OPEN INPUT

* END DCB OPENS
TIME
ST R1,JGYYDDDD SAVE JULIAN DATE
BAL RBAL,JULGREG CONVERT TO MM/YY/DD
MVC HEADER(L'HEAD),HEAD INITIALIZE HEADER
MVC HEADER+L'HEAD(L'HEADER-1),HEADER+L'HEADER-1 CLEAR
MVC DDONAME,INPDDN MOVE IEBCOPY JCL FILE NAME
BAL RBAL,GETNAMES GET SELECTION DSN
ZAP PAGES,-P'1' INITIALIZE PAGE COUNT
MVC HEADDATE,JGMMDDYY MOVE MM/YY/DD TO HEADING
BAL RBAL,HEADPAGE PRINT PAGE HEADER

MAINLOOP GET INPUT,INAREA READ INPUT RECORD
CLI INAREA,C'-' SEPARATOR LINE
BNE MAINOK NO
CLC =C'SEPARATE',PARM 'SEPARATE' PARM?
BNE MAINNOTS NO
CP PAGES,-P'1' FIRST PAGE?
BNE MAINNOTS NO
BAL RBAL,HEADPAGE EJECT TO NEW PAGE

MAINNOTS MVC LINE+(SCALE-SUBHEAD)(80),SCALE SET SCALE
B MAINPR GO PRINT LINE
MAINOK CLC INMEM,-BC' ' MEMBER NAME PRESENT?
BNE MAINRFMT YES
MVC LMEM,-BC'** SET FLAG
MVC LCOUNT,-C'NOTE:' SET NOTE INDICATOR
B MAINMVC

MAINRFMT MVC LMEM,INMEM SET MEMBER NAME
MVC LCOUNT,INCOUNT SET RECORD NUMBER
MVC L7380,IN7380 MOVE COLUMNS 73-80
MAINMVC MVC LSOURC,INSOURC MOVE COLUMNS 1-72
MAINPR BAL RBAL,PRINT GO PRINT LINE
B MAINLOOP CONTINUE UNTIL E-O-F
MAINEOF DS 0H
PUT PRINTER,SUBHEAD PRINT FOOTER

* BEGIN DCB CLOSE
MVC PRCLosl(PRCLoslN),CLOSED INITIALIZE CLOSE LIST
CLOSE (PRINTER),MF=(E,PRCLosl) CLOSE IT

* MVC IPCLosl(IPCLoslN),CLOSED SET INPUT CLOSE LIST
CLOSE (INPUT),MF=(E,IPCLosl) CLOSE INPUT

* END DCB CLOSE

© 1997. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (940) 455 7050, fax (940) 455 2492.
END00          LA      R15,0            SET COMPLETION CODE 00
ST      R15,COMPCODE            INTO STORAGE
B       ENDING                GO TO ENDING

* EJECT

***********************************************************************
*** LINKAGE CONVENTIONS EXITING PROGRAM ***
***********************************************************************
ENDING      L      R14,COMPCODE    R14 SAVES COMP CODE
LR      R1,R13        R1 SAVES ADDR OF MY S.A.
L      R13,4(Ø,R1)    R13 RESTORED, PTR CALLER S.A.
FREEMAIN  R,LV=WORKDLEN,A=(R1) FREE MY SAVE/WORK AREA
LR      R15,R14       R15 SET TO COMP CODE
LM      RØ,R12,2Ø(R13) RØ-R12 RESTORED
L      R14,12(Ø,R13)   R14 RESTORED
MVI  12(R13),X'FF' SET COMPLETION SIGNAL
BR      R14           RETURN TO CALLER

* BEGIN STUB DEFINE
EJECT

***********************************************************************
*** GET JOB AND PDS DSN NAMES ***
***********************************************************************
* THANKS TO MR. MARK HOFFMAN FOR THIS LOGIC
***********************************************************************

GETNAMES      ST    RBAL,SAVGNBAL    SAVE LINKAGE REGISTER

* XR      R15,R15        ADDRESS OF PSA
USING      PSA,R15       ESTABLISH ADDRESSABILITY
L      R14,FLCCVT        ADDRESS OF CVT
DROP      R15           DROP ADDRESSABILITY TO PSA
USING      CVTMAP,R14    ESTABLISH ADDRESSABILITY TO CVT
L      R15,CVTTCBP      ADDRESS OF NEXT TCB POINTER
L      R15,4(Ø,R15)     ADDRESS OF CURRENT TCB
DROP      R14           DROP ADDRESSABILITY TO CVT
USING      TCB,R15       ESTABLISH ADDRESSABILITY CURRENT TCB
L      R14,TIOELNGH     ADDRESS OF TIO
USING      TIOENTRY,R15  ESTABLISH ADDRESSABILITY TO TIO
MVC  HEADJOBN,TIOCNJOB  MOVE JOB NAME TO HEADER
MVC  HEADJOBN-4(4),-C'JOB-' SET JOBNAME ID

* DROP      R15          DROP ADDRESSABILITY TO TCB
LA      R15,TIOELNGH    ADDRESS OF FIRST TIO ENTRY
DROP      R14           DROP ADDRESSABILITY (HLASM OBJECTS)
USING      TIOENTRY,R15  ESTABLISH ADDRESSABILITY TO TIO
GNTIOTLP  CLI  TIOELNGH, X'00' END OF TIO CHAIN?
BE      GNRETURN        YES (SHOULDN'T HAPPEN)
CLC  TIOEDDMM(Ø),DDNAME PDS NAME FOUND?
BE      GNDSN           YES
XR      RØ,RØ          CLEAR REGISTER
IC      RØ,TIOELNGH    INSERT ENTRY LENGTH
AR      R15,RØ         POINT TO NEXT ENTRY

**B** GNTIOTLP CONTINUE

GNDSN XR R1,R1 CLEAR REGISTER
ICM R1,7,TIOEJFCB ADDRESS OF JFCB
USING JFCB,R1 ESTABLISH ADDRESSABILITY TO JFCB
MVC HEADDNSN,JFCBDSNM MOVE DSN TO HEADER
MVC HEADDNSN-4(4),=C'DSN=' SET DSN ID IN HEADER
DROP R1,R15 DROP ADDRESSING TO JFCB,TIOT,ENTRY
GNRETURN L RBAL,SAVGNBAL RESTORE LINKAGE REGISTER
BR RBAL RETURN
EJECT

******************************************************************************
*** CONVERT JULIAN DATE TO GREGORIAN DATE ***
******************************************************************************
*
JULGREG ST RBAL,SAVJGBAL SAVE LINKAGE REGISTER
ZAP JGDAYS,JGYYDDD+2(2) SAVE DAYS FROM BEGINNING OF YEAR
ZAP JGMONTHS,=P'1' INITIALIZE MONTH
LA R15,JANUARY LOAD ADDRESS OF DAYS/MONTH TABLE
LA 0,L'JANUARY ... WIDTH OF TABLE
LA 1,DECEMBER ... END OF TABLE
ZAP FEBRUARY,=P'20' SET NON LEAP YEAR DAYS
CLC =X'2000',JGYYDDD YEAR 20XX?
BE JGYR2000 YES
JG20THCN TM JGYYDDD+1,1 LEAP YEAR?
BO JGLOOP NO
TM JGYYDDD+1,X'12'
BM JGLOOP NO
JGYR2000 AP FEBRUARY,=P'1' ADJUST
JGLOOP CP JGDAYS,0(L'JANUARY,R15) CURRENT MONTH?
BNH JGFOUND YES
AP JGMONTHS,=P'1' INCREMENT MONTH
SP JGDAYS,0(L'JANUARY,R15) DECREMENT DAYS PER CURRENT MONTH
BXLE R15,R0,JGLOOP CONTINUE
JGFOUND UNPK JGMMDYY(2),JGMONTHS UNPACK MONTH
UNPK JGMMDYY+3(2),JGDAYS UNPACK DAY
UNPK JGMMDYY+6(3),JGYYDDD+1(2) UNPACK YEAR
MVI JGMMDYY+2,C'/ ' SEPARATE MONTH AND DAY
MVI JGMMDYY+5,C'/ ' SEPARATE DAY AND YEAR
OI JGMMDYY+1,C'0' FORCE MONTH NUMERIC
OI JGMMDYY+4,C'0' FORCE DAY NUMERIC
OI JGMMDYY+7,C'0' FORCE YEAR NUMERIC
JGRETURN L RBAL,SAVJGBAL LOAD LINKAGE REGISTER
BR RBAL RETURN
*
END STUB DEFINE
EJECT

******************************************************************************
*** PRINT ROUTINE ***
******************************************************************************
*
PRINT PUT PRINTER,LINE PRINT LINE
MVI LINE,C' ' SET SEED
**MVC LINE+1(L'LINE), LINE CLEAR LINE**

**DOUBLES**

**BCTR**

**PUT**

**HEADPAGE MVC**

**ED**

**AP**

**ED**

**LA**

**MVI**

**BR**

**EJECT**

***********************************************************************

*** FIXED DATA AREA ***

***********************************************************************

**HEAD DC**

**SUBHEAD DC**

**ORG**

**DC**

**SCALE DC**

**DC**

**ORG**

**OPEND**

**CLOSED**

* BEGIN DCB CONSTANTS

**PRINTERD DCB**

**INPUTD DCB**

**INPDDN EQU**

* END DCB CONSTANTS

**JGMO**

* END CONSTANTS

**MOVEPARM MVC**

**LTORG**

**EJECT**

***********************************************************************

*** DSECT FOR MY SAVE AREA AND VARIABLES. ***

***********************************************************************

**WORKD DSECT**

**MYSAVE DS**

**COMPCODE DS**

**RETCDE DS**

**RISAVE DS**

**PAGES DS**

**DOUBLE DS**

**DDNAME DS**

**PARM DS**

* BEGIN STUB LINK SAVE

**SAVGN**

**SAVJG**

* END STUB LINK SAVE
* BEGIN OPEN/CLOSE LIST
     DS ØD
PROPENL OPEN (,),MF=L
PROPENLN EQU *-PROPENL
PRCLOSLL CLOSE (,),MF=L
PRCLOSLN EQU *-PRCLOSLL
IPOPENL OPEN (,),MF=L
IPOPENLN EQU *-IPOPENL
IPCLOSLL CLOSE (,),MF=L
IPCLOSLN EQU *-IPCLOSLL
* END OPEN/CLOSE LIST
* BEGIN DCB DSECTS
PRINTER DCB DDNAME=PRINTER,DEVD=DA,DSORG=PS,LRECL=133,
       BLKSIZE=133,MACRF=(PM),RECFM=FBA
PRINTERL EQU *-PRINTER
INPUT DCB DDNAME=INPUT,DSORG=PS,MACRF=GM,EODAD=MAINEOF
INPUTL EQU *-INPUT
* END DCB DSECTS
JGMOTBL DS PL2'0'
JANUARY DS P'31'
*  M A M J J A S  O N
FEBRUARY DS P'28,31,30,31,30,31,30,31,31,30,31,30,31,30'
DECEMBER DS P'31'
JGDAYS DS PL2
JGMONTHS DS PL2
JGMMDDYY DC 'MM/DD/YY'
JGYYDDD DS F
* END DSECT INSERT
HEADER DS CL133
ORG HEADER+L'HEAD+Ø
HEADJOBN DS CL8,' C' DSN='
HEADDSN DS CL44,5C
HEADDATE DS CL8
ORG HEADER+L'HEADER-5
PAGENO DS CL4
ORG
INAREA DS CL93
ORG INAREA
INSOURC DS CL72
INMEM DS CL8
IN738Ø DS CL8
INCOUNT DS CL5
ORG
LINE DS CL133
ORG LINE+1
LMEM DS CL8,' C
LCOUNT DS CL5,' C
LSOURC DS CL72
L738Ø DS CL8
ORG
DS ØD

© 1997. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (940) 455 7050, fax (940) 455 2492.
Simulating Include files in REXX

THE PROBLEMS

The purpose of this article is to explain a process I have developed for simulating include files in REXX EXECs. One of the accepted ways to prevent repetition of code in any language is to use include files for the common code. In this way the code is part of the program and is included in it at compile time. In REXX there is no such feature.

The accepted procedure is to use external REXX EXECs and to invoke them as subroutines or functions. The drawback to this solution is that only values passed as parameters on the call are available to the called subroutine (or function). If it was defined internally within the REXX EXEC then all the caller’s values would be accessible unless a PROCEDURE command included in the subroutine.

A number of problems are encountered with parameter passing and returning when calling external REXX EXECs. The main ones are:

- It is not possible to pass a list of variables based on stems. In this case it would be necessary to pass each value as a separate parameter.
• The number of parameters that can be passed is limited to 30 (or 15 – depending on the REXX PTF level). Although this seems to be a reasonable number there are a number of cases where this is not sufficient.

• It is possible to pass more than one value in a single parameter (separated by blanks, for example), however this does not work if blanks are to be included in the parameter value itself.

• Any change in the parameters required by the called REXX requires changes to each EXEC that invokes it.

• It is only possible to return one value from the called EXEC. This value is returned as the parameter of the return statement and is available in the variable RESULT (when the EXEC is called as a subroutine) or as the function return value (when called as a function).

POSSIBLE SOLUTIONS
A number of options are available to solve these problems. However, none of these options covers all possibilities.

• Pass and return the values via the stack. This is done by using PUSH and PULL commands. It is advisable to use the NEWS TACK command before filling the stack and the DELSTACK after reading it so as to hide the contents of other stacks from the EXEC.

This solution works quite well although it is a bit messy in the code. It will not work if the external EXEC is invoked as a TSO or ISPF command. In this case the lines queued by the invoked EXEC will be interpreted by the operating system as commands. To prevent this it is necessary to add a NEWSTACK command after filling the stack before returning to the caller and then a DELSTACK in the caller before reading the values from the stack. For example:

Test:

'NEWSTACK'
queue var1
queue var2
call testcall
pull result_value_1
pull result_value_2
'DELSTACK'

Testcall:
pull var1
pull var2
...
...
queue result_value_1
queue result_value_2

The main disadvantage of this method is that the order of the caller and called must be maintained.

- Similar to the previous but, so as to solve the problem of the order of values, queue actual commands to set values and the INTERPRET them after reading them from the stack. For example, to pass the values of variables A and B to the called EXEC:

  Caller:
  queue "a = " a
  queue "b = " b

  Called:
  do queued()
  pull line
  interpret line
  end

  The called EXEC would return values to the caller in the same way. This solution has the added advantage that passing of stem based values is easier.

- Pass the values using ISPF commands VPUT and VGET. This solution is similar to the previous one except that the values are stored in ISPF controlled variables. The main disadvantage of this solution is the limited length of names of variables in ISPF (8 characters). Furthermore, the passing of stem–based variables is almost impossible via this method.

Pass and return the values as a single value separated by blanks (as given above). On return a PARSE command would be used to separate the result into its variables. This will solve the problem of the name lengths and is much clearer in the code. However, if values contain blanks, this would not work. It would be possible to use a different character but the same problem would arise if that character exists in one of the values. For example:
We were left looking for a solution that would have the same effect as an include statement in PL/I etc. In this way the code would be included in the main EXEC and all the variables would be accessible. The solution we found was to use the INTERPRET command, so as to execute commands inline within the EXEC. This interpret command allows the construction of commands in REXX variables and execution of these commands as if they were part of the code. In this way it is possible to build dynamic commands within the EXEC.

The solution was to construct the required code externally to the main EXECs. These external EXECs are then read in at the start of the ISPF application and constructed in a single variable, which contains all the commands that were in the original EXECs.

Whenever it is necessary to execute the commands, an INTERPRET command on the variable is performed. In this way all the variables are fully accessible. Furthermore, any changes made to the EXEC are automatically reflected in the caller and no change is needed so as to pass the extra parameters. The only stipulation is that these external EXECs can only use values that are available in all the EXECs.

The constructed command variables are stored as ISPF variables and can be retrieved by any EXEC that requires to execute them. The best way to perform this, we found, was to construct one more ISPF variable that contains all the VGET commands for all the command variables. In this way, if a new EXEC is added, then no change is needed. This is especially important since the INTERPRETed commands can themselves include INTERPRET commands.
/* This REXX EXEC is used for a creating a line of commands that can */
/* be used by another REXX EXEC in an INTERPRET command. */
/* */
/* The EXEC will read the lines of the specified file and return them */
/* as a single variable with a semi-colon between the lines. */
/* The calling EXEC can then execute the commands using the INTERPRET */
/* command. */
/* */
/* The EXEC is useful where it is necessary to execute the same */
/* commands in a number of EXEC but it is not possible to put them in */
/* in a called EXEC. For example, when the function must changed a */
/* number of variables. */
/* */
/* In this way, any change will be reflected in all the EXECs. */
/* */
/* The EXEC receives the following parameters: */
/* */
/* 1. A list of libraries to search for the member. */
/* 2. Name of the member to fetch. */
/* */
arg libraries, member.
address TSO
/* Search the libraries looking for the member. If it is not found */
/* then exit with no string. */
do i = 1 to words(libraries)
    filename = "'"word(libraries,i)"("member")'"
    if sysdsn(filename) = 'OK' then leave
end
if i > words(libraries) then return ''
/* Read in all the lines of the exec. */
ALLOC F(EXEC) DS("filename") REUSE SHR
'EXECIO * DISKR EXEC ( STEM LINES. FINIS'
'FREE F(EXEC)"
/* Now loop over all the lines concatenating them into one string. */
/* Insert a semi-colon between the commands. */
/* If the last character of the line is a comma then the next line is */
/* a continuation. In this case the trailing comma is removed and the */
/* lines are concatenated. */
all_lines -
do 1 - 1 to lines.0
    line - strip(lines.i)
    if right(line,1) = ',' then
do
        line - left(line,length(line)-1)
        all_lines - all_lines||line
    end
else
    all_lines - all_lines||line';'
end

**********************************************************************/
/* Now return the result to the caller so that it can be used in an */
/* INTERPRET command. */
**********************************************************************/
return all_lines

Below is an EXEC that builds all the ISPF variables for the commands. Each one contains the code from one EXEC:

**********************************************************************/
/* This EXEC is used to set up the internal macros for the CSP41 */
/* EXECs. It is invoked at the entry to CSP41. */
**********************************************************************/
search_libraries = CSP4slib()
parse var search_libraries sysexecl sysexec2
if sysexec2 = '' then sysexec2 = sysexc1

CSP4CHKP = cparsmem(search_libraries , 'CSP4CHKP')
CSP4CHMS = cparsmem(search_libraries , 'CSP4CHMS')
CSP4DETL = cparsmem(search_libraries , 'CSP4DETL')
CSP4EFIL = cparsmem(search_libraries , 'CSP4EFIL')
CSP4QUAL = cparsmem(search_libraries , 'CSP4QUAL')
CSP4SLST = cparsmem(search_libraries , 'CSP4SLST')
CSP4VGET = cparsmem(search_libraries , 'CSP4VGET')
CSP4VPUT = cparsmem(search_libraries , 'CSP4VPUT')
address ISP ExEC ,
"VPUT (CSP4CHKP,CSP4CHMS,CSP4DETL,CSP4EFIL"
   "CSP4QUAL,CSP4SLST,CSP4VGET,CSP4VPUT) SHARED"
CSP4MGET = 'address ISP ExEC',
   '"VGET (CSP4CHKP,CSP4CHMS,CSP4DETL,CSP4EFIL,'
   "CSP4QUAL,CSP4SLST,CSP4VGET,CSP4VPUT) SHARED"
address ISP ExEC 'VPUT (CSP4MGET,SYSEXECl,SYSEEXEC2) SHARED'
exit

© 1997. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (940) 455 7050, fax (940) 455 2492.
It is also possible to use the function directly by using the interpret command on the result of the call to the external function PARSEMEM. For example:

```
Interpret parsemem('LIB1 LIB2','MEMBER')
```

Below is an example of an EXEC that will be interpreted:

```
******/*
*******
*******
***********/
/* This EXEC is used by the EXECs to set the qualifiers for the temp files. */
******/*
parse value time() with hh':' mm ':' ss .
scndqual = 'T'||hh||mm||ss
qual = mdr||p||'.'||scndqual
```

Following is an example the use of the EXECs in another EXEC:

```
/* REXX */
*/---------------*/
/* C.S.P. rel. 4.1 - UTILITIES */
/* */
/* This program generate a job that move a member from one msl */
/* to another. The program can get as input an asterisk (*) as */
/* a wildcard character to represent one or more characters in */
/* the member name. */
/* To move 2 or more members, put the names in a file and use the */
/* file options. */
/* */
/* Libraries : Panels - SYS.ALL.ISRPLIB */
/* Skels - SYS.ALL.ISPSLIB */
/*_Msgs - SYS.ALL.ISPMLIB */
/* Macros - SYS.CSP.EXEC */
/* */
address ISPEXEC
*/---------------*/
/* Get the command for GETting all the commands from the ISPF */
/* variables. Execute it to get all the commands. */
/* */
/* Next exec the VGET EXEC commands so as to get all the variables */
/* needed for the EXEC from the application profile pool. */
/* */
'VGET CSP4MGGET'
interpret CSP4MGGET
interpret CSP4VGET
function = 'COPYMEM'

/* Display panel */
/* */
"DISPLAY PANEL(CSP4M2M)"
Ret = Rc

do while Ret ≠ 8
    call process_first_screen
    "DISPLAY PANEL(CSP4M2M)"
    Ret = rc
end
exit

process_first_screen:
Csrfield = ''
Error = FALSE

/*-----------------------------------------------*/
/* Checking the data in the screen */
/* Checking if the files exist ... */
/*-----------------------------------------------*/

if Sysdsn("""FROMMSL"""") = "OK" then
    "SETMSG MSG(CSP410G)"
    Csrfield = "FROMMSL"
    return
end

if Sysdsn("""TOMSL"""") = "OK" then
    "SETMSG MSG(CSP410G)"
    Csrfield = "TOMSL"
    return
end

/*-----------------------------------------------*/
/* Generate qualifiers for temporary files. Use pre-built command */
/*-----------------------------------------------*/
p = ..
interpret CSP4QUAL
/*-----------------------------------------------*/
/* Edit file if needed */
/*-----------------------------------------------*/
interpret CSP4EFIL
/*-----------------------------------------------*/
/* Moving the csp commands to the temp dsn. */
/*-----------------------------------------------*/
address ISPEXEC "TBCREATE CSP4M2M NAMES(LINE) NOWRITE"
address TSO "NEWSTACK"
do i = 1 to memb.0
    Line = "LIST MEMBER(" || STRIP(MEMB.I) || ") "
    "TBADD CSP4M2M"
    Line = "PRINT(Y) OUTFILE(TEMP) MSL(FROMMSL) REFTYPE(*);"
    "TBADD CSP4M2M"
end
end
/*-----------------------------------------------*/
/* Handling the list associates option.          */
/*-----------------------------------------------*/
if Lsta = 'Y' then
  do
    Line = "LISTA INFILE(TEMP) PRINT(Y) OUTFILE(TEMP1);"
    "TBADD CSP4M2M"
    Line = "MSL M(TOMSL) ROMSL(FROMMSL);"
    "TBADD CSP4M2M"
    Line = "COPYLIST INFILE(TEMP1) PRINT(Y) REPLACE(Y);"
    "TBADD CSP4M2M"
  end
else
  do
    Line = "MSL M(TOMSL) ROMSL(FROMMSL);"
    "TBADD CSP4M2M"
    Line = "COPYLIST INFILE(TEMP1) PRINT(Y) REPLACE(Y);"
    "TBADD CSP4M2M"
  end
/*-----------------------------*/
/* Creating the skeleton file.              */
/*-----------------------------------------------*/
"FTOPEN TEMP"
"VGET (ZTEMPF)"
call csp4jobc mem.l , 'CMEM'
"FTINCL CSP4M2M"
"FTCLOSE"
"TBCLOSE CSP4M2M"
"TBERASE CSP4M2M"
/*-----------------------------------------------*/
/* Checking if automatic submission or editing the job is */
/* wanted.                                              */
/*-----------------------------------------------*/
if Edit = 'Y' then
  "EDIT DATASET(''|ZTEMPF||')"
else
  address TSO "SUBMIT ''|ZTEMPF||'"
interpret CSP4VPUT
return

The interpreted commands CSP4QUAL, CSP4VGET, and CSP4VPUT are used in all the EXECs in the system. In this way if, for example, we wish to change the structure of the temporary files prefix, then it is sufficient to make the change in CSP4QUAL and there is no need to make changes to every EXEC.
NOTES ABOUT THE INTERPRET COMMAND

The following points should be noted when building the EXECs:

• Interpret commands can be nested. So it is possible to include in the EXECs built calls to other EXECs via interpret commands.

• All loops must be complete within the command string. It is not possible to include only the first part of the loop in the interpreted string and to have part of the loop outside of it.

• Any signal command will cause immediate exit from the interpret command. Labels are permitted within the string but are ignored.

• It is not possible to jump into the middle of an interpret command string.

• Any subroutine or function calls in the interpreted string will not search for the label within the string. Labels will be searched for only in the EXEC itself. However, after the subroutine/function completes, control is returned to the interpret command at the point where the call occurred.

This last point allows the possibility to build generic functions that can invoke specific subroutines to perform certain tasks. In this way, an EXEC that supplies a general structure for a series of actions can be defined. Within this interpreted EXEC it is possible to include call commands to perform specific tasks required by the EXECs that include the interpret command. The interpret command will invoke the local subroutines whilst maintaining the general structure of the EXEC. The local subroutines will perform the EXEC-specific commands and then return control to the interpret command.

An example of this would be a generic structure for building jobs via ISPF screens. The structure of the main loop could be maintained in one interpreted EXEC with calls to subroutines that perform the DISPLAY commands for the panels and the FTINCL commands for the skeleton construction.

Take the above code as an example. All the code from the start of the skeleton building to the end is standard in all EXECs. The only section that is different is the includes. All that needs to be done is to take that section and create another interpreted EXEC. In place of the FTINCL command a call command would be inserted. This would call a subroutine included in the main EXEC and would be different in each EXEC.
OVERHEADS

There are a number of overheads inherent in this method. These are:

- The call to PARSEMEM to set-up each EXEC into the variables at the start and the VPUT commands to save them. This step can be particularly heavy especially if there are many EXECs.

- The VGET commands to get the variables with the commands within them.

- Commands included in an interpret command execute slower than commands in the actual code. This is because the command has to be parsed every time whereas the standard EXEC commands are parsed only once.

- The EXEC cannot include any SIGL or internal calls. This increases the complexity of the EXEC.

These overheads must be weighed against the gains in productivity in future updates. The load time can be reduced by loading only those EXECs that are actually used. They can be loaded at first-use time and, in this way, only those EXECs used will be loaded.

One way of doing this is to set up the variable that is to contain the EXEC so as to self load the EXEC. For example:

```
CSP4QUAL = "CSP4QUAL~PARSEMEM('LIB1 LIB2','CSP4QUAL');", 
"VPUT CSP4QUAL; INTERPRET CSP4QUAL"
```

This would then be saved as the value of CSP4QUAL. When it is INTERPRETed the first time it will simply parse the same named EXEC and replace the stored string with the created one. It then INTERPRETs the new string. In future calls to the EXEC the newly created string will be used.

Jonathan Blitz
Senior Systems Programmer
AnyKey Computer Systems Ltd (Israel)

© Xephon 1997
Organize your disks and claim free space

Do you ever need to move files from one volume to another in a fast and clean way? Do you ever wonder why user X likes to allocate one cylinder to create a ten-line file, instead of allocating one track? If you do, you may find something of interest below.

IBM supplies a utility program with MVS known as ADRDSSU. In its standard form, it is not very user-friendly. However, thanks to Mike Cowlishaw, we can easily overcome that handicap and make it work for our benefit by designing REXX programs around it. This is what I have done with the following two programs.

The first program, MOVEFILE, is designed around the COPY option of ADRDSSU, and allows you to move a file or a group of files between volumes. Simply invoke the MOVEFILE EXEC, passing as argument the name of the file you want to move. The EXEC will ask you the original volume of the file and the destination volume. With those three arguments, the EXEC creates and submits a job that will perform the operation. Since the file is going to be freshly allocated, ADRDSSU allows you to specify how you want it to be allocated – in blocks, tracks, or cylinders. Personally, I prefer tracks, and so, as a side-effect of the move operation, those cylinder mammoths to which I was referring previously will be reduced to more decent proportions.

If you develop the MOVE concept, you can use it to downsize the allocated space, and then put the file back in its original volume. That is what the second program, REALLOC, does. REALLOC is simply a double MOVE, where the destination volume functions as a temporary volume. REALLOC generates a two-step job – the first moves the file to another volume of your choice, and the second puts it back in the original place.

USAGE NOTES
Both MOVEFILE and REALLOC are especially useful to deal with a group of files. They can be VSAM, SEqs, or PDS. To specify a group of files, use the ADRDSSU filtering rules (see DFSMSdss Storage Administration Reference). As a reminder of those rules, here are some examples:
IBM.*  Means any file with only two qualifiers, the first being IBM.

IBM.**  Means any file with any number of qualifiers, the first being IBM.

IBM*.** Means any file with any number of qualifiers, the first beginning with IBM.

If a file that is to be processed is allocated by another task, it will not be processed. The same is true for an empty PDS. If such is the case, a return code of 8 or 4 will appear. You may ignore it, since all the other files are correctly processed.

VSAM files will not be space-reduced, so REALLOC is useless for them. If you use REALLOC for a group of files, be sure that the temporary volume you specify does not contain any file that fits into your generic specification, otherwise they will be moved in the jobs second step. As an example, if you REALLOC IBM.* files in volume A, using volume B as temporary volume, and volume B also contains IBM.* files, they will all end up in volume A.

MOVEFILE

```rexx
/* REXX MVS **********                       */
/* MoveFile - Moves a file or group of files */
/* from one volume to another */
/* ********************* */

jobfile = userid() || ".movefile"  /* job file */
xx = msg(off)  /* check if jobfile */
"free da("jobfile")"  /* already exists */
okay = sysdsn(jobfile)  /* if not, create it*/
if okay="OK" then do
  "free da("jobfile")"
  "alloc da("jobfile") dd(ddtemp),
    new reuse blksize(3200) lrecl(80),
    recfm(f,b) dsorg(ps) space(1 1) tracks"
  if rc = 0 then do
    say "Error" rc " allocating "jobfile
    signal saida
  end
else
  "alloc da("jobfile") dd(ddtemp) shr"
  /* retrieve previous */
  if rc = 0 then do
    /* volume to use */
    say "Error" rc " allocating "jobfile
  end

```
signal saida
end
execio 5 diskr ddtemp
do 5
  pull linha
end
parse var linha . "DS(INCLUDE(" dsnll "))"
execio 1 diskr ddtemp
parse pull linha . "(" dsnll ")"
execio 1 diskr ddtemp "(finis"
pull linha . "(" dVol 22 ")"
end
arg dsnl . /* get arg (filename)*/
if dsnl → "" then do /* get its volume */
  dsnll = dsnl
  xx = listdsi(dsnl)
  voll = sysvolume
end
say "MoveFile: Input File? ( ENTER for" dsnll
pull dsnl .
if dsnl = "" then dsnl = dsnll
say " Input Volume? ( ENTER for" voll
pull voll .
if voll = "" then voll = voll1
say " Output Volume? ( ENTER for" vol22
pull vol2 .
if vol2 = "" then vol2 = vol22
dropbuf
dsn1 = strip(dsn1,"")
queue "/"userid()"0 JOB MSGCLASS=X,MSGLEVEL=(1,1)"
queue "/"STEP1 EXEC PGM=ADRDSSU,REGION=2M"
queue "/"SYSPRINT DD SYSOUT=*"
queue "/"SYSIN DD *
queue " COPY DS(INCLUDE("dsn1")) -"
queue " INDYNAM ("voll") -"
queue " OUTDYNAM ("vol2") -"
queue " CATALOG -"
queue " DELETE -"
queue " FORCE -"
queue " TGTALLOC (TRK) -"
queue " PROCESS (SYS1)"
queue "/*"
queue "***"
"execio * diskw ddtemp (finis"
"submit "jobfile"
saida:
"free da("jobfile")"
"free dd(ddtemp)"
exit

REALLOC
/* REXX MVS ******************************************************/
Realloc - Reallocates a file in tracks

jobfile = userid() || "realloc"   /* job file */
xx = msg(off)   /* check if jobfile already exists */
"free da("jobfile")"   /* if not, create it */
okay = sysdsn(jobfile)   /* if not, create it */
if okay="OK" then do
   "free da("jobfile")"
   "alloc da("jobfile") dd(ddtemp),
      new reuse blksize(3200) lrecl(80),
      recfm(f,b) dsorg(ps) space(1) tracks"
   if rc = 0 then do
      say "Error" rc " allocating "jobfile
      signal saida
   end
else do   /* If jobfile exists. */
   "alloc da("jobfile") dd(ddtemp) shr"   /* retrieve previous */
   if rc = 0 then do   /* volume to use */
      say "Error" rc " allocating "jobfile   /* as default */
      signal saida
   end
execio 5 diskr ddtemp
do 5
   pull linha
end
parse var linha . "DS(INCLUDE(" dsnl ))"
execio 1 diskr ddtemp
parse pull linha . "(" volll ")" .
execio 1 diskr ddtemp "(finis"
pull pull linha . "(" vol22 ")" .
end
arg dsnl .   /* get arg (filename)*/
if dsnl = "" then do   /* get its volume */
   dsnl = dsnl
   xx = listdsi(dsnl)
   volll = sysvolume
end
say"Realloc: Input File? ( ENTER for" dsnl
pull dsnl .
if dsnl = "" then dsnl = dsnl
say" Input Volume? ( ENTER for" volll
pull volll .
if volll = "" then volll = volll
say" Temporary Volume? ( ENTER for" vol22
pull vol2 .
if vol2 = "" then vol2 = vol22
dropbuf
dsnl = strip(dsnl,"")
queue "/"userid()"0 JOB MSGCLASS=X,MSGLEVEL=(1,1)"
queue "/"STEP1 EXEC PGM=ADRDSU,REGION=2M"
queue "/"SYSPRINT DD SYSOUT=*"

Useful Assembler macros – part 3

We complete our look at the Assembler macros BSM31, BALRXA, and CALLXA. Also included are AUTHON and AUTHOFF which will dynamically turn on/off authorization through the traditional authorization SVC.

BSM31 MACRO
* SET ADDRESSING MODE TO 31 BIT IF RUNNING UNDER XA/ESA
* NEUTRAL UNDER MVS/370
USES WORK REGISTER, DEFAULT TO R15
WORKREGISTER CAN BE OVERWRITTEN BY BSM (RX)
WORK REG POINTS TO NEXT INSTR AND CONTAINS ADDR MODE
CODE FOR SUPPORT OF NON-XA (MVS/370) WILL ONLY BE GENERATED IF
GLOBAL VARIABLE FROM INITR &MVS370S-SUP IS SPECIFIED OR &SPLEVEL=1;
IF MACRO INITR IS NOT USED AND &SPLEVEL > 1, IT IS STILL POSSIBLE
TO FORCE GENERATION OF MVS/370 VIA THE PARAMETER MVS370=SUP.
CODE FOR SUPPORT OF XA/ESA WILL ONLY BE GENERATED IF &SPLEVEL > 1.

MACRO
&NAME BSM31 &REG,&MVS370-NOTSUP
GBLC &MVS370S COMES FROM INITR IF THIS MACRO IS USED
GBLC &SYSSPLV MACRO LEVEL
SLEVEL TEST SET SYSSPLV
LCLC &NONXA

&NONXA SETC 'B31'.&SYSNDX'
AIF ('&MVS370S' NE '').INTSUPP

&MVS370S SETC '&MVS370' . SET ONLY FROM PARAMETER IF INITR IS NOT USED
INTSUPP ANOP
AIF ('&MVS370S' EQ 'NOTSUP').SUPP
AIF ('&MVS370S' EQ 'SUP').SUPP
MNOTE 8, 'MVS370 MUST BE INDICATED AS NOTSUP OR SUP'
MEXIT

SUPP ANOP
AIF ('&SYSSPLV' GT '1').XASupp XA-MACRO LEVEL
&MVS370S SETC 'SUP' FORCE MVS370 SUPPORT

XASupp ANOP
AIF ('&REGR' EQ '').Rnull
AIF ('&REGR'(1,1) EQ '').Areg
AGO .Rnull

Areg ANOP
&REGR SETC '&REGR(1)'
AGO .REG

Rnull ANOP
&REGR SETC '15'

Reg ANOP
&NAME DS 0H.
AIF ('&MVS370S' EQ 'NOTSUP').XA
AIF ('&SYSSPLV' LT '2').NONXA BYPASS IF NOT XA/ESA MACLEVEL

Testxa (&Aregr)
LTR &REGR,&REGR . TEST FOR MODE
BP &NONXA . MVS/370

XA ANOP
LA &REGR,&NONXA . POINT TO AMODE 31 CODE
0 &REGR,&NONXA-4 TURN ON AMODE 31 BIT
BSM 0.&REGR . BRANCH TO AMODE 31 CODE
Cnop 0.4 ALIGN
DC X'80000000' AMODE 31 BIT

&NONXA DS 0H.

NONXA ANOP
Balr &REGR,0 LET WORK REG POINT TO NEXT
Mexit
Mend

BALRXA MACRO

* GENERATES BASSM RX,RY IF RUNNING UNDER XA/ESA, CALL AS BALRXA R14,R15
* GENERATES BALR RX,RY IF RUNNING UNDER MVS/370, CALL AS BALRXA R14,R15
* ENSURES THAT A SUBROUTINE IN AN XA/ESA ENVIRONMENT IS CALLED IN RIGHT
* ADDRESSING MODE; THE REQUIREMENT IS THAT R15 CONTAINS CORRECT
* ADDRESSING MODE IN HIGH ORDER BIT; THE ADDRESSING MODE OF A SUB-
* ROUTINE IS RETURNED TO THE USER FROM THE LOAD MACRO.
* CODE FOR SUPPORT OF NON-XA (MVS/370) WILL ONLY BE GENERATED IF
* GLOBAL VARIABLE FROM INTR &MVS370S=SUP IS SPECIFIED OR &SPLEVEL=1;
* IF MACRO INTR IS NOT USED AND &SPLEVEL > 1, IT IS STILL POSSIBLE
* TO FORCE GENERATION OF MVS/370 VIA THE PARAMETER MVS370=SUP.
* CODE FOR SUPPORT OF XA/ESA WILL ONLY BE GENERATED IF &SPLEVEL > 1.
* IF SUBROUTINE RETURNS IN DIFFERENT ADDRESSING MODE THAN IT WAS
* CALLED, THEN ADDRESSING MODE IS CORRECTED BACK.

MACRO

&NAME BALRXA &RREG,&BREG,&MVS370=NOTSUP
GBLC &MVS370S COMES FROM INTR IF THIS MACRO IS USED
GBLC &SYSSPVLV MACRO LEVEL
SLEVEL TEST SET SYSSPVLV
LCLC &XA24,&XA31
LCLC &NEXTOP
	
&XA24 SETC 'BL1'.&SYSNDX'
&XA31 SETC 'BL2'.&SYSNDX'
&NEXTOP SETC 'BL3'.&SYSNDX'
AIF ('&MVS370S' NE ').INTSUPP
&SYS370S SETC '&MVS370'. SET ONLY FROM PARAMETER IF INTR IS NOT USED
INTSUPP ANOP
AIF ('&MVS370S' EQ 'NOTSUP').SUPP
AIF ('&MVS370S' EQ 'SUP').SUPP
MNOTE 8, 'MVS370 MUST BE INDICATED AS NOTSUP OR SUP'
MEXIT
SUPP ANOP
AIF ('&SYSSPVLV' GT '1').XASUPP XA-MACRO LEVEL
&MVS370S SETC 'SUP' FORCE MVS370 SUPPORT
XASUPP ANOP
AIF ('&SYSSPVLV' LT '2').NONXA BYPASS IF NOT XA/ESA MACLEVEL
TESTXA (&RREG) .
LTR &RREG,&RREG . TEST FOR XA
BM &XA31 . USE BASSM FOR XA/ESA 31-BIT
BZ &XA24 . USE BASSM FOR XA/ESA 24 BIT
AIF ('&MVS370S' EQ 'NOTSUP').XA
NONXA ANOP
BALR &RREG,&BREG . LINK
AIF ('&SYSSPVLV' LT '2').BYPNON2 BYPASS IF NOT XA/ESA MACLEVEL
B &NEXTOP NEXT INLINE INSTRUCTION
AGO .XA
BYPNON2 ANOP
MEXIT
XA ANOP
&XA24 DS 0H
BASSM &RREG,&BREG . LINK
BSM24 (&RREG) . ENSURE STILL IN 24 BIT MODE

© 1997. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (940) 455 7050, fax (940) 455 2492.
B &NEXTOP NEXT INLINE INSTRUCTION

&XA31 DS $0H
BASSM &RREG,&BREG . LINK
BSM31 (&RREG) . ENSURE STILL IN 31 BIT MODE

&NEXTOP DS $0H
BALR &RREG,0 . LET RET-REG CONTAIN SAME VALUE AS IF REAL BALR
MEND

CALLXA MACRO

* WORKS AS CALL MACRO AT THE SAME TIME AS ENSURING CORRECT ADDR-MODE
* GENERATES BASSM 14,15 IF RUNNING UNDER XA/ESA.
* GENERATES BALR 14,15 IF RUNNING UNDER MVS/370.
* ENSURES THAT A SUBROUTINE IN AN XA/ESA ENVIRONMENT IS CALLED IN RIGHT
* ADDRESSING MODE; THE REQUIREMENT IS THAT R15 CONTAINS CORRECT
* ADDRESSING MODE IN HIGH ORDER BIT; THE ADDRESSING MODE OF A SUB-
* ROUTINE IS RETURNED TO THE USER FROM THE LOAD MACRO.
* CODE FOR SUPPORT OF NON-XA (MVS/370) WILL ONLY BE GENERATED IF
* GLOBAL VARIABLE FROM INITR &MVS370S=SUP IS SPECIFIED OR &SLEVEL=1;
* IF MACRO INITR IS NOT USED AND &SLEVEL > 1, IT IS STILL POSSIBLE
* TO FORCE GENERATION OF MVS/370 VIA THE PARAMETER MVS370=SUP.
* CODE FOR SUPPORT OF XA/ESA WILL ONLY BE GENERATED IF &SLEVEL > 1.
* IF SUBROUTINE RETURNS IN DIFFERENT ADDRESSING MODE THAN IT WAS
* CALLED, THEN ADDRESSING MODE IS CORRECTED BACK.

MACRO

&NAME CALLXA &ENTRY,&OPRNDs,&VLPARa,&ID=.&MF=I,&MVS370=NOTSUP
GBLB &IHBSWA,&IHBSWB
GBLC &IHBNO
LCLC &GNAME
GBLC &MVS370S COMES FROM INITR IF THIS MACRO IS USED
GBLC &SYSSPLV MACRO LEVEL
SLEVEL TEST SET SYSSPLV
LCLC &XA24,&XA31
LCLC &NEXTOP

&A24 SETC 'CX1'.&SYSNDX'
&A31 SETC 'CX2'.&SYSNDX'
&NEXTOP SETC 'CX3'.&SYSNDX'
AIF ('&MVS370S' NE '').INTSUPP
&MVS370S SETC '&MVS370'. SET ONLY FROM PARAMETER IF INITR IS NOT USED
INTSUPP ANOP
AIF ('&MVS370S' EQ 'NOTSUP').SUPP
AIF ('&MVS370S' EQ 'SUP').SUPP
MNOTE 8,'MVS370 MUST BE INDICATED AS NOTSUP OR SUP'
MEXIT
SUPP ANOP
AIF ('&SYSSPLV' GT '1').XASUPP XA-MACRO LEVEL
&MVS370S SETC 'SUP' FORCE MVS370 SUPPORT
XASUPP ANOP
&IHBNO SETC '309'
&GNAME SETC 'IHB'.&SYSNDX'
&IHBSWA SETB ('&VLPARA' EQ 'VL')
&IHBSWB SETB ('&ENTRY' EQ '(15)')
AIF ('&VLPARA' NE '' AND '&VLPARA' NE 'VL').ERROR4
AIF ('&MF' EQ 'L' AND '&ENTRY' NE '' ).ERROR1
AIF ('&MF' EQ 'L' AND '&ID' NE '').ERROR2
AIF ('&MF' NE 'L' AND '&ENTRY' EQ '').ERROR3
AIF ('&MF' EQ 'L').CONTC
AIF (&IHBSWB).CONTCC
CNOP 0.4

&NAME B +8 BRANCH AROUND VCON
&GNAME.B DC V(&ENTRY) ENTRY POINT ADDRESS
CONTC AIF ('&OPRnds' EQ '' AND X
('&MF' EQ 'I' OR '&MF' EQ 'L')).CONTB
CONTA IHBOPLTX &ENTRY,&OPRnds,&NAME,MF-MF
CONTB AIF ('&MF' EQ 'L').EXITI
AIF (&IHBSWB).CONTD
L 15,&GNAME.B LOAD 15 WITH ENTRY ADR
CONTD ANOP
AIF ('&SYSSPLV' LT '2').NONX BYPASS IF NOT XA/ESA MACLEVEL
TESTXA (14) .
LTR 14,14 . TEST FOR XA
BM &XA31 . USE BASSM FOR XA/ESA 31-BIT
BZ &XA24 . USE BASSM FOR XA/ESA 24 BIT
AIF ('&MVS370S' EQ 'NOTSUP').XA

NONXA ANOP
BALR 14,15 . LINK
AIF ('&SYSSPLV' LT '2').BYPNON2 BYPASS IF NOT XA/ESA MACLEVEL
B &NEXTOP NEXT INLINE INSTRUCTION

XA ANOP
&XA24 DS 0H
BASSM 14,15 . LINK
BSM24 (14) . ENSURE STILL IN 24 BIT MODE
B &NEXTOP NEXT INLINE INSTRUCTION

&XA31 DS 0H
BASSM 14,15 . LINK
BSM31 (14) . ENSURE STILL IN 31 BIT MODE

&NEXTOP DS 0H
BYPNON2 ANOP
AIF ('&ID' EQ '').EXITX
DC X'4700' NOP INSTRUCTION WITH
DC AL2(&ID) ID IN LAST TWO BYTES
DS 0H
EXITX ANOP
BALR 14,0 . LET RET-REG CONTAIN SAME VALUE AS IF REAL BALR
EXITI MEXIT
CONTC ANOP
&NAME DS 0H AGO .CONTCC
ERROR1 IHBERMAC 73,&IHBN0,&ENTRY ENTRY W/ MF=L
MEXIT
ERROR2 IHBERMAC 74,&IHBN0,&ID ID W/ MF=L
MEXIT
ERROR3 IHBERMAC 26,&IHBN0 ENTRY SYMBOL MISSING

© 1997. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (940) 455 7050, fax (940) 455 2492.
AUTHON MACRO

* AUTHON TURNS ON AUTHORIZATION IF NOT ALREADY ON
* PARAMETER BRANCH=YES CALL TESTAUTH WITH BRANCH=ENTRY; DEFAULT NO
MACRO

&NAME AUTHOR &BRANCH=NO
&APFON SETC 'AO1'.'&SYSNDX'
&NAME DS 0H
LR R0,R5
SAVE R15
AIF ('&BRANCH' EQ 'YES').BRANCH
TESTAUTH FCNT=1 TEST FOR APF

BRANCH ANOP
AIF ('&BRANCH' NE 'YES').NBRANCH
TESTAUTH FCNT=1,BRANCH=NO TEST FOR APF

&APFON DS 0H .
MEND

AUTHOFF MACRO

* AUTHOFF TURNS OFF AUTHORIZATION IF NOT ALREADY OFF
* PARAMETER BRANCH=YES CALL TESTAUTH WITH BRANCH=ENTRY; DEFAULT NO
MACRO

&NAME AUTHOFF &BRANCH=NO
&APFOFF SETC 'AF1'.'&SYSNDX'
&NAME DS 0H
LR R0,R15
SAVE R15
AIF ('&BRANCH' EQ 'YES').BRANCH
TESTAUTH FCNT=1 TEST FOR APF

BRANCH ANOP
AIF ('&BRANCH' NE 'YES').NBRANCH
TESTAUTH FCNT=1,BRANCH=NO TEST FOR APF

&APFOFF DS 0H .
MEND

Nils Plum
Systems Programmer (Denmark) © Xephot 1997

MacKinney Systems has announced JES Queue Client for Printers. The utility is a VTAM-based print management system which prints any report from the JES output queue to network attached printers defined to VTAM. Printer types supported are SNA, non-SNA, and SCS. Reports in the JES output queue are automatically selected based on their DESTID and printed to the printer defined for that destination. Both machine code and ASA control characters are supported.

For further information contact:
MacKinney Systems, 2740 S Glenstone, Suite 103, Springfield, Missouri, 65804-3737, USA.
Tel: (417) 882 8012
Fax: (417) 882 7569.

Advent Software Corporation has announced SysStat for MVS Release 2.2.0. The utility provides OS/390 conversion support and an enhanced user interface. New features include the HSM Query and Command facility (HSM/QCF), which aids management of DFSMSHsm resources in the TSO/ISPF and batch environments. Users can search DFSMS databases to retrieve migrated and back-up dataset statistics, and review HSM volume control information.

For further information contact:
Advent Software Corporation, 340 W Butterfield Road, Suite 4B, Elmhurst, IL 60126, USA.
Tel: (630) 297 5449
Fax: (630) 941 7980.

IBM has announced a replacement for its IMSPARS and IMSASAP IMS tuning products for MVS, adding a range of new capabilities and features. IMS Performance Analyser, available now, will provide the reporting tools of the older products and have an ISPF CUA user interface for report requests. It will also provide for revised and enhanced reports, as well as brand new reports, and will support IMS Versions 4, 5, and 6 from a single LOADDLIB. There will be an option for using GDDM for selected graphical reports, and an ability to save selected report data for PC tools.

Contact your local IBM marketing representative for further information.

Boole & Babbage have announced enhanced capabilities for Command MQ. Command MQ now supports end-to-end availability management for Microsoft Message Queuing Server ( MSMQ). The utility which supports MVS provides a centralized console for managing IBM’s MQSeries and MSMQ and overseeing the primary areas of their operations in distributed environments.

For further information contact:
Boole & Babbage, 3131 Zanker Road, San Jose, CA 95134 - 1933, USA.
Tel: (408) 526 9000
Fax: (408) 526 3053 or
Boole & Babbage (UK) Ltd, Burnham House, Clivemont Road, Maidenhead, SL6 7RU, UK.
Tel: (01628) 771909
Fax: (01628) 770458.