220

MVS update

January 2005

In this issue

A xephon PUBLICATION

Published by

Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

Phone: 214-340-5690
Fax: 214-341-7081

Editor

Trevor Eddolls
E-mail: trevore@xephon.com

Publisher

Colin Smith
E-mail: info@xephon.com

Subscriptions and back-issues

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

MVS Update on-line

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

Contributions

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

Printed in England.

TRAP2 and RP illustration

This program illustrates the use of TRAP2 and RP (Resume Program) instructions.

TRAP2 is a simple instruction that gives control to a routine in the current context of execution (AR/GR/PSW) and returns control wherever you want (next to TRAP2 or with PSW alteration), with modified or unmodified AR and GR. You have to make sure that the CPU is in the primary-space or AR mode, and bit 47 of DUCT is on.

The following program runs a loop (eight times with BCT) and each time issues a TRAP2. TRAP2 gives control to a TRAP program, which prints R0, R1, R2, and PSW contents. And also, if the TRAP program finds that R2 contains F'2', it overwrites the instruction pointed to by the PSW (BCT) with a X'0101' (PR) to end the loop.

This TRAP/RP pair can easily be used to design an on-line disassembler or debugger. Please read the comments in the code and refer to the *Principles of Operation* manual for more sophisticated code with TRAP and RP.

CODE

```
MYTRAP2 CSECT
MYTRAP2 AMODE 31
MYTRAP2 RMODE 24
*=================================================================*
* MODULE-NAME     : MYTRAP2                                       *
* DESCRIPTIVE-NAME: Shows TRAP/RP processing                      *
* REQUIREMENTS    :  APF                                          *
* FUNCTION        :                                               *
* Initialize TRAP environment, and issue an 8-times loop with     *
*    a coded TRAP2 instruction.                                   *
* The TRAP  takes control, displays GRØ-GR2 and PSW contents      *
* and resumes the program.                                        *
* The TRAP program also checks the content of GR2, if GR2=2 then  *
* it overwrites instruction (BCT) pointed to by PSW with a PR to  *
* end the main routine.                                           *
*=================================================================*
```

```
         BAKR  R14,Ø
         LR    R12,R15
         USING MYTRAP2,R12
* Begin: Prepare TRAP2 environement
         L     R1,PSATOLD-PSA(,Ø)                Current TCB
         L     R1,TCBSTCB-TCB(,R1)               Get STCB
         L     R5,STCBDUCV-STCB(,R1)             Get DUCT in R5
         MODESET MODE=SUP,KEY=ZERO               Get into Key Ø
         LA    R7,TRPR                           Load TRAP CB addr
         O     R7,=XL4'ØØØØØØØ1'                 Bit E (31) = on
         ST    R7,44(,R5)                        Store CB in DUCT
* Main Routine Loop
         LA    R2,8                              Loop limit
LOOP     EQU   *
         M     RØ,=F'2'                          Multiply R1 by 2
         LA    R15,98                            Just for test
         TRAP2                                   TRAP
         BCT   R2,LOOP                           Loop
         B     EXIT                              Exit
         CNOP  Ø,8
*================================================================*
* Tran Control Block                                             *
* Offset                                                         *
* X'ØØ'     =>    'xxxxxxxxxxxxxPRxxxxxxxxxxxxxxxx'              *
*  Bit 13 and 14 controls the use of current PSW bit 31          *
*                          the use of current PSW bit 12 and 33-127*
*         if P=X'ØØ' Current PSWbit31 must be zero               *
*                    Current PSWbit12 will be set to one         *
*                    Current PSWbit97-127 will be stored in      *
*                            PSWfield33-63                       *
*                    Current PSWbit33-96  will not be stored in  *
*                            PSWfield and zero are put in        *
*                            PSWfield64-127                      *
*         if P=X'Ø1' Current PSWbit31 can be zero or one.        *
*                    Current PSWbit12 will be store in           *
*                            PSWField12                          *
*                    Current PSWbit64-127 will be stored in      *
*                            PSWfieldbit64-127                   *
*         if R=X'ØØ' GRs bit32-63 stored in four-bytes location  *
*                    from GR field (GRs bitØ-31 not stored)      *
*         if R=X'Ø1' GRs bitØØ-63 stored in eightbytes location  *
*                    from GR field .                             *
*================================================================*
TRPR     DS    ØF
         DC    BL4'ØØØØØØØØØØØØØØ1ØØØØØØØØØØØØØØØØØ'  Bit P=1,R=Ø
         DC    F'Ø',F'Ø'                         Reserved
TRSA     DC    A(SA)                             Save area Addr
         DC    F'Ø'                              Reserved
TRPGM    DC    A(PGM)                            Trap program addr
         DC    8F'Ø'                             Reserved
```

```
         CNOP  Ø,4
*=================================================================*
* Trap program  :                                                 *
* Display GRØ,1,2 and PSW contents                                *
* If R2 =2 it updates the PSW pointed instruction (so BCT )       *
* with a PR to end program.                                       *
*                                                                 *
* Chaining      :                                                 *
* TCB->STCB->DUCT=> DUCT+X'44' => Trap Control Block Addr(TRCB)   *
*                   TCRB+X'ØC' => Trap Save Area addr(TRSA)       *
*                   TRSA+X'1Ø' => PSW  128bits                    *
*                                 Instruction to   after TRAP     *
*                   TRSA+X'2Ø' => GRØØ-15                         *
* Resume Program:                                                 *
* RP parmlist layout  Ø-12                                        *
*                   X'13'Bit P => Specify PSW size Ø=8  1=16      *
*                   X 14'Bit R => if Ø, GR field is 4bytes        *
*                              => if 1 and Bit15=Ø GR field is    *
*                                 8bytes                          *
*                              => if 1 and Bit15=1 in both        *
*                                 GR field and field 2.           *
*                   X'15'Bit D => see above                       *
*                   X'1Ø'Offset to psw fld                        *
*                   X'2Ø'Offset to AR  fld                        *
*                   X'3Ø'Offset to GR  fld                        *
*                   X'4Ø'Offset to GR  fld additional (see b15)   *
*=================================================================*
PGM      EQU   *
         OPEN  (SYSTRAP,OUTPUT)                    Open
         L     R1,PSATOLD-PSA(,Ø)                  Current TCB
         L     R1,TCBSTCB-TCB(,R1)                 Get STCB
         L     R5,STCBDUCV-STCB(,R1)               Get DUCT in R5
         L     R5,44(,R5)                          Get TRAP2 CB
         BCTR  R5,Ø                                Align on DW
         L     R4,12(,R5)                          Get TRAP2 TRSA
         CLC   X'28'(4,R4),=F'2'                   If R2=2 instr at
         BNE   NOUPDT                              PSW (BCT) is overw
         L     R3,X'1C'(R4)                        ritten with (PR) to
         MVC   Ø(2,R3),=XL2'Ø1Ø1'                  end main loop
         MVC   MSGGRØØ(16),=CL16'chg instr at psw'
NOUPDT   EQU   *
* Edit GRs and PSW
         UNPK  MSGGR+4(9),X'2Ø'(5,R4)              Unpk RØ
         TR    MSGGR+4(8),TAB                      Edit RØ
         MVI   MSGGR+12,X'4Ø'                      Clear
         UNPK  MSGGR+13(9),X'24'(5,R4)             Unpk R1
         TR    MSGGR+13(8),TAB                     Edit R1
         MVI   MSGGR+21,X'4Ø'                      Clear
         UNPK  MSGGR+22(9),X'28'(5,R4)             Unpk R2
         TR    MSGGR+22(8),TAB                     Edit R2
```

```
         MVI   MSGGR+3Ø,X'4Ø'                        Clear
         UNPK  MSGGR+31(9),X'1Ø'(5,R4)               Unpk PSW Ø-31
         TR    MSGGR+31(8),TAB                       edit PSW
         MVI   MSGGR+39,X'4Ø'                        clear
         UNPK  MSGGR+4Ø(9),X'1C'(5,R4)               Unpk PSW 64-127
         TR    MSGGR+4Ø(8),TAB                       edit PSW
         MVI   MSGGR+48,X'4Ø'                        clear
         PUT   SYSTRAP,MSGGRØØ                       Print
* Prepare Resume Program (RP)
         MVC   PPSW(16),16(R4)                       Return PSW
         MVC   GR1(4),92(R4)                         Restore R15
         LA    R15,PARM                              Load Parm for RP
         LM    RØ,R14,32(R4)                         Restore register
         RP    Ø(R15)                                Resume program
PARM     DC    B'ØØØØØØØØØØØØØ1ØØ'                   Bit P=1 R=Ø D=Ø
         DC    Y(PPSW-PARM)                          16-31 offset to psw
         DC    Y(AR1-PARM)                           32-47 offset to AR
         DC    Y(GR1-PARM)                           48-63 Replace R15
*                                                    (GR b2 of RP instr)
PPSW     DC    2D'Ø'                                 RP PSW
AR1      DC    F'Ø'                                  RP AR
GR1      DC    F'Ø'                                  RP GR
         DC    F'Ø'                                  RP GR Addtionnal
         CNOP  Ø,4
MSGGRØØ  DC    CL16'GRØ GR1 GR2 PSW:'
MSGGR    DC    CL2ØØ' '
SYSTRAP    DCB DDNAME=SYSTRAP,MACRF=PM,DSORG=PS,LRECL=2ØØ,RECFM=FB
         CNOP  Ø,4
TAB      DC    15XL16'ØØ'
         DC    C'Ø123456789ABCDEF'
EXIT     EQU   *
         XR    R15,R15
         PR                                    return to MVS
         CNOP  Ø,8
SA       DS   64F
RØ       EQU   Ø
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R1Ø      EQU   1Ø
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
```

```
R15      EQU   15
         IHAPSA
         IHASTCB
         IKJTCB
         END
```


Edit macro to build ADRDSSU control card input

The following is a very simple ISPF edit macro that will format a list of dataset names and add the necessary formatting to allow them to be used as input to a standard IBM DSS move job.

At the installation where I work, I have to regularly produce lists of DB2 datasets that we then must move to new volumes.

We normally use FDREPORT actually to PUNCH the dataset names to a standard PS dataset for volumes where the DB2 datasets reside.

Below is the JCL for the FDREPORT job. It will look at all our volumes that start with DB2 and list all datasets on those volumes.

```
//JXB7884R    JOB   (JXB),'J.BRADLEY',CLASS=A,NOTIFY=JXB7884
//*
//*      ************************************************************
//*      *                                                          *
//*      * FDRREPORT TO PUNCH DATASET NAMES FOR INCLUSION IN       *
//*      * FDRCOPY.                                                 *
//*      *                                                          *
//*      ************************************************************
//*
//STEP1       EXEC  PGM=FDREPORT,REGION=2M
//SYSUDUMP    DD    SYSOUT=*
//SYSPRINT    DD    SYSOUT=*
//ABRMAP      DD    SYSOUT=*
//ABRSUM      DD    SYSOUT=*
```

```
//SYSPUNCH   DD    DSN=JXB7884.SYSIN2,DISP=SHR
//PUNMASK    DD    *
  <DSN>
/*
//SYSIN      DD    *
  SELECT VOLG=DB2
  PUNCH FDRLIB=PUNMASK
  PRINT RPTYPE=SELPCH
```

Once created, the dataset JXB7884.SYSIN2 will contain all the dataset names to be moved. It will also contain the ***VTOC and SYS1.VTOCIX dataset names for each volume processed and the SYS1.VVDS that pertains to these volumes.

A subset of the contents of JXB7884.SYSIN2 is shown below:

```
    ****VTOC
  DB2.DSNDBD.DH2L.XM0.I0001.A001
  DB2.DSNDBD.DH5I.XM2.I0001.A032
  DB2.DSNDBD.DH2A.ST.I0001.A006
  DB2.DSNDBD.DH2I.XLM0.I0001.A001
  SYS1.VTOCIX.DB2001
  SYS1.VVDS.VDB2001
```

The edit macro, MKCARDS, runs against the JXB7884.SYSIN2 dataset. It performs the following functions:

1 Removes any lines containing the word VTOC.

2 Removes any lines containing the word VVDS.

3 Changes the DB2 dataspace names to the VSAM cluster names.

4 Inserts the DSS start INCLUDE card as line 1.

5 Suffixes all lines with a DSS continuation character of ‘-’.

6 Inserts closing bracket for the DSS control cards as the last line of the dataset.

On completion, the input will be formatted like this:

```
INC (                                                          -
  DB2.DSNDBC.DH2L.XM0.I0001.A001                               -
  DB2.DSNDBC.DH5I.XM2.I0001.A032                               -
  DB2.DSNDBC.DH2A.ST.I0001.A006                                -
  DB2.DSNDBC.DH2I.XLM0.I0001.A001                              -
 )
```

This dataset can then be used as input to the DSS JCL shown below:

```
//JXB7884D    JOB   (JXB),'J.BRADLEY',CLASS=L
//*
//*       ************************************************************
//*      *                                                            *
//*      * DSS VERSION OF FDRMOVE.                                    *
//*      *                                                            *
//*      * CONTROL CARD INPUT FROM DATASET                            *
//*      *                                                            *
//*      * LOGINDDNAME(DISK1) -                                       *
//*      * SELECTMULTI(ANY) -                                         *
//*      *                                                            *
//*       ************************************************************
//*
//SØ1Ø         EXEC  PGM=ADRDSSU
//SYSPRINT     DD     SYSOUT=*
//FILTERDS     DD     DSN=JXB7884.SYSIN2,
//                      DISP=SHR
//SYSIN        DD      *
 COPY DATASET(FILTERDD(FILTERDS)) -
      CANCELERROR -
      CATALOG -
      DELETE
/*
```

The MKCARDS edit macro looks like:

```
/* REXX */

/*       ************************************************************/
/*       *                                                          */
/*       * MACRO NAME:  MKCARDS                                     */
/*       *                                                          */
/*       * PURPOSE:     TO FORMAT FDREPORT OUTPUT TO EXCLUDE        */
/*       *              ***VTOC, SYS1.VTOCIX, AND SYS1, VVDS        */
/*       *              DATASETS.                                   */
/*       *                                                          */
/*       *              THEN TO RENAME ANY DB2 DATA COMPONENT       */
/*       *              NAMES TO CLUSTER NAME.                      */
/*       *                                                          */
/*       *              THEN ADRDSSU STATEMENTS ARE ADDED SO FILE   */
/*       *              CAN BE USED AS INPUT TO A DSS MOVE JOB.     */
/*       *                                                          */
/*       * PROGRAMMER:  JOHN BRADLEY.                               */
/*       *                                                          */
/*       ************************************************************/

VARINC = "'INC ( '"          /* SETUP VARIABLE FOR 1ST LINE CONTENT.*/
```

```
VARBRK = "' )'"                 /* SETUP VARIABLE FOR LAST LINE.       */

ADDRESS "ISPEXEC"               /* WORK UNDER ISPF.                    */
ISREDIT MACRO                   /* ISPF EDIT MACRO.                    */
ADDRESS "ISREDIT"               /* EDIT COMMANDS FOLLOW.               */

 "EX ALL"                       /* EXCLUDE ALL INPUT.                  */
 "FIND VTOC ALL"                /* FIND THE WORD VTOC IN DATA.         */
 "FIND VVDS ALL"                /* FIND WORD VVDS.                     */
 "DEL ALL NX"                   /* DELETE ALL LINES THAT WERE FOUND.   */
 "RESET"                        /* DISPLAY WHAT IS LEFT.               */
 "CHG DSNDBD DSNDBC ALL"        /* CHANGE DB2 NAMES TO BE CLUSTER.     */
 "LINE_AFTER Ø =" VARINC        /* INSERT DSS INC STATEMENT AT TOP.    */
 "(FIRST) = LINENUM .Zfirst"    /* Set up pointer to first line.       */
 "(LAST) = LINENUM .ZLast"      /* Point to last line.                 */

 DO WHILE FIRST <= LAST         /* Loop until all lines processed.     */
                                /* nEXT LINE suffixes with a -         */
   ADDRESS "ISREDIT" "LINE" first "= LINE + <69 '-'>"
   FIRST = FIRST + 1            /* Process next line.                  */
 END                            /* End of loop.                        */

ADDRESS "ISREDIT"               /* EDIT COMMANDS FOLLOW.               */
 "LINE_AFTER .ZLAST =" VARBRK /*INSERT CLOSING BRACKET ON LAST LINE.*/

EXIT Ø                          /* Leave the macro with RC= Ø.         */
```

Elizabeth Bradley
Systems Programmer
Meerkat Computer Services Ltd (UK)

zAAP – zSeries Application Assist Processor

IBM has introduced a Java co-processor – or what is called 'zSeries Application Assist Processor (zAAP)' – that operates asynchronously to execute Java programs under IBM JVM. Similar to Integrated Facility for Linux (IFL), which is the dedicated processor for Linux, IBM offers attractive pricing for this zAAP dedicated for Java load.

zAAP basically provides a cost-effective specialized z/OS

Java execution environment. This article aims to provide a brief overview of zAAP and what the potential benefits are.

HOW DOES zAAP WORK?

zAAP can be configured so that it operates within the CPs within the logical partitions running z/OS. zAAPs operate asynchronously with the general processors to execute Java programming under the control of the IBM Java Virtual Machine (JVM), thereby reducing the demands and capacity requirements on the general purpose CPs.

Execution of the JVM processing cycles on a zAAP is handled as a function of the IBM Software Developer's Kit (SDK) 1.4 product, z/OS 1.6, and the Processor Resource/Systems Manager (PR/SM).

The JVM instructs z/OS to redirect its Java processing cycles to the zAAP. On completion of the Java processing cycle or when a non-Java function is to be executed by the application, control is redirected back to the general CP.

The key point to be noted is that the Java application(s) doesn't have to be modified to exploit this benefit of running the JVM processing cycles in zAAPs. The switch to a zAAP from general purpose CP for zAAP-qualifying work and the switch back from a zAAP to a general-purpose processor when non-qualifying work is encountered are completely transparent to the application.

Another point to be noted is that this Java processor, or zAAP, does not run z/OS, z/VM, or Linux, so it cannot be used to do any other work. Hence unlike other processors (CP, ICF, IFL) zAAP can do nothing on its own and cannot be IPLed.

zAAP AND COST REDUCTION

zAAPs cost (at around US$125K per zAAP) significantly less than the general CPs, and provide an attractive option for customers who want to run their Java load on a mainframe.

The maintenance price for the zAAPs is also significantly lower than that of general purpose CPs and similar to the maintenance price for IFLs.

All Java-based applications, including WebSphere application servers, can execute on these lower-cost zAAPs.

zAAP doesn't carry an MSU rating and IBM doesn't include this processing capacity for computing software charges.

Additional processing power can be exclusively added for processing the ever-changing Java-based Internet load by increasing zAAPs without affecting the MSU rating. The number of zAAPs that can be ordered is limited by the number of permanently purchased CPs and, of course, by the number of available engines on a given machine model.

Even for existing loads, the sub-capacity option is expected to benefit when zAAP is used for handling the Java load. This is because Java work processed by zAAP is outside the normal processor time collection schemes.

BENEFITS OF zAAP

zAAP provides an attractive option for customers who would like to seamlessly integrate their Java-based Internet applications with their core legacy applications but have been deterred by the cost factor.

Off-loading the Java workload to zAAP can help reduce the demands and capacity requirements on general purpose CPs.

zAAPs do not provide performance improvements for the Java and WebSphere workloads at this point in time (though there is an indication that it is being considered and may possibly be available in the future). The Java application being deployed in the same LPAR as the associated database – as against multiple physical servers – can lead to the following advantages:

- Tightly-integrated and highly-secure environments.
- Simplified server infrastructure and maintenance.
- Improved operational efficiencies by reducing the number of TCP/IP programming stacks, firewalls, and physical interconnections.
- Reduced processing latencies by means of closer access to the database and reduced network processing.

PREREQUISITES FOR zAAP

Introduced in April 2004 along with z890 mainframes, zAAP will be supported in z890, z990, and future models only. The operating system has to be upgraded to zOS/zOSe 1.6. IBM SDK for z/OS, Java 2 Technology Edition, V1.4 with PTF (or later) is also required.

IMS Versions 7, 8, or 9, DB2 Versions 7 or 8, and CICS 2.3 can exploit zAAP.

WebSphere Version 5.1 or above can exploit zAAP. IBM also offers some relaxation for WAS load to exploit zAAP before moving to z/OS 1.6

CONTROLLING THE USE OF zAAP

IBM SDK for z/OS, Java 2 Technology Edition, V1.4 includes the following new run-time options for zAAP support:

- *-Xifa:on* – this can enable Java work to be run on the zAAP if the zAAPs are available. This setting is assumed by default.
- *-Xifa:off* – this is designed to disable the use of zAAP.
- *-Xifa:projectn* – this is designed to estimate projected zAAP usage and write this information to STDOUT at intervals of *n* minutes.

- *-Xifa:force* – this is designed to force Java to continue attempting to use zAAP, even if none are available. It would typically be specified for the purpose of collecting RMF/SMF data to assess potential zAAP and applicable only to z/OS 1.6

Note that the command line options that start with –X refer to non-standard Java interpreter options and are anticipated to be unique to IBM.

zAAP-ELIGIBLE WORK ON STANDARD CP

z/OS 1.6 provides two new options, IFACrossOver and IFAHonorPriority, that can be defined in the IEAOPT*xx* member of SYS1.PARMLIB. They control how zAAP-eligible work is switched between standard processors and zAAPs.

The following gives a summarized view of the CrossOver parameter.

IFACrossOver = Yes (default) allows crossover, permitting Java work to compete for standard CP resources in addition to executing on zAAPs. It is used when the preference is to fully utilize the cheaper zAAP resource. Its disadvantage is that it prevents the execution of Java work in standard CP even if it is available and zAAPs are fully utilized.

IFACrossOver = No means the standard processor may not run any zAAP-eligible work unless there is no zAAP operating in the LPAR. It is used when the flexibility of using the standard CP for Java is required. Its disadvantage is that tracking work is complex and the cost of Java processing is higher.

When crossover is permitted, the amount of Java crossing over onto the standard CPs can be controlled by the HonorPriority setting.

With *IFAHonorPriority = Yes* (default), WLM will manage the priority of zAAP-eligible work for standard processors. It can be used when the Java work has to be dispatched ahead of any lower priority non-Java work.

With *IFAHonorPriority* = *No*, Java work can only use available CP capacity if there is no other non-Java work waiting to execute. It can be used when the Java workload is to use standard CP only when there is no other workload.

zAAP PROJECTION TOOL

According to IBM, 'While zAAPs are capable of executing up to 100% of Java cycles, the reality is that most applications are not 100% Java'. It is estimated that around 50% to 70% of the actual Java workload will be off-loaded to zAAP.

The amount of Java application code executed by zAAP(s) is dependent on the amount of Java cycles used by the relevant application(s) and on the zAAP execution mode selected. Hence, the amount of general purpose processor saved and in turn the cost benefits are expected to vary depending on the customer.

To allow the customers to get an idea of the potential savings they can achieve by running their Java application in zAAP, IBM has also announced the zAAP Projection Tool. This tool basically collects the usage information of how much CPU time is used in executing Java code and categorizes it as eligible for the zAAP or not eligible. It is designed so that only code written in Java and Java system native code is enabled to run on a zAAP.

You can run a Java workload representative of your production load and, through the Java log, this tool will report how much of the workload would potentially be shipped to zAAP.

The tool runs under Java 2 Technology Edition, SDK 1.3.1, and hence can be used with lower versions of software than required for exploiting zAAP – for example with WAS 5.0 and CICS 2.2.

This tool is useful in determining whether it is worth moving towards zAAP and, if so, the number of zAAPs that would form the optimal configuration for your workload.

A spreadsheet summarization tool is also available to assist in the analysis of the zAAP Projection Tool output. The zAAP Projection Tool workbook can:

- Combine data from multiple JVMs.
- Combine data from multiple address spaces, service classes, and LPARs (eg Websphere uses multiple address spaces each producing a Java log that needs to be combined to arrive at meaningful information).
- Combine the data and align to intervals such as the RMF interval used.
- Adjust zAAP utilization factoring in z/OS configuration (controlling use of zAAPs).

The tool, along with the Excel sheet, can be downloaded from http://www-1.ibm.com/servers/eserver/zseries/zaap/gettingstarted.

Note that z/OS 1.6 will have functions to include zAAP capacity planning information in SMF/RMF records.

Sasirekha Cota
Tata Consultancy Services (India)

Expanding CA-Endevor compressed listings

Our installation uses the CA-Endevor software package from Computer Associates to control the application development and promotion process. When all of our LPARs were contained on one physical machine (CEC in IBM terms), shared DASD sufficed to make all elements owned by Endevor available to both our production and test LPARs. For the purpose of integrity, we decided to split our LPARs onto separate machines and into separate sysplex environments. This necessitated

the requirement to ship elements from the test LPARs, where the development occurs, to the production LPARs, where the applications actually run. One of the elements that is shipped is the compile and link-edit listings that are stored by Endevor. These are shipped so that if there is a production abend the applications programmer can have the compile listing available for debugging purposes. The compile listings are stored in a compressed format, to save space. Endevor provides a utility to read and expand the listing to human-readable format. When we decided to make separate sysplexes, however, we elected to license the use of Endevor only on our test sysplex. This meant the listings that were shipped over to the production sysplex would not be usable by the applications programmers for debugging, because we were not licensed to run the Endevor utility to expand the compressed listings on our production sysplex. We decided to examine the listings and found that we could easily write a utility to expand the listings back to human-readable format ourselves, without the use of any Endevor utility. This is the mechanism we decided to use on the production sysplex.

It turns out that the compression method used by Endevor is the replacement of repeating character strings with either a 2- or a 3-byte flag string. We found two different types of flag string. The first was for the replacement of repeating blanks with a 2-byte field. The first byte was a flag indicator byte containing X'FD', followed by another 1-byte field that was the repetition count. The second was for the replacement of repeating non-blank characters with a 3-byte field. The first byte was a flag indicator byte containing X'FC', followed by a 1-byte field containing the character that had been compressed, followed by another 1-byte field containing the repetition count.

With the above information, a rather simple REXX EXEC was written to read a compressed listing file stored as a PDS member, and to expand the flag fields back to their original format. If the EXEC is called under ISPF, then ISPF browse is invoked to display the listing, otherwise the user is given the

name of the dataset where the expanded listing was stored so they can view it by other means.

NDVRLIST REXX

```
/*                              rexx comment *** start standard header
NDVRLIST Scan Endevor list library and expand listing to human readable
                               rexx comment *** end   standard header */
parse upper arg sys subsys mem
ds = "ENDEVOR.PROD."sys"."subsys".LIST("mem")" /* name dataset/member */
"ALLOC DD(X@X) DA('"ds"') SHR REUSE"
"EXECIO * DISKR X@X (STEM LINE. FINIS"
"FREE DD(X@X)"
do i = 1 to line.Ø
   pass = Ø
   do until pos("FD"x,line.i) = Ø & pos("FC"x,line.i) = Ø
      pass = pass + 1
      fdpos = pos("FD"x,line.i)
      fcpos = pos("FC"x,line.i)
      select
         when fcpos > Ø then do                  /* expand characters */
              fccount = x2d(c2x(substr(line.i,fcpos+1,1)))
              fcdata  = substr(line.i,fcpos+2,1)
              line.i = delstr(line.i,fcpos,3)
              line.i = insert(fcdata,line.i,fcpos-1,fccount,fcdata)
              end
         when fdpos > Ø then do                     /* expand blanks */
              fdcount = x2d(c2x(substr(line.i,fdpos+1,1)))
              line.i = delstr(line.i,fdpos,2)
              line.i = insert(" ",line.i,fdpos-1,fdcount," ")
              end
         otherwise nop                        /* leave the line as is */
      end /* select */
   end /* do until */
end i

tmp = userid()".D"date(J)".T"time(S)
"ALLOC DD(C1PRINT) DA('"Tmp".LIST')",
      "REUSE NEW SPACE(1Ø,5) CYL ",
      "RELEASE UNIT(VIO) LRECL(133) BLKSIZE(2793Ø) RECFM(F B A)"
"EXECIO" line.Ø "DISKW C1PRINT (STEM LINE. FINIS"
dsn="'"Tmp".LIST'"
x = listdsi(dsn)
"FREE DD(C1PRINT)"
if sysused = Ø then do
   say "Program "pgm" not found in ENDEVOR.PROD."sys"."subsys".LIST" ,
       ", please try again..."
   return 12
```

```
   end
if sysvar("SYSISPF") = "NOT ACTIVE" then
   say "The expanded listing has been stored in dataset" dsn
else address ISPEXEC "BROWSE DATASET("dsn")"
exit
```

Brett Berger
Systems Programmer (USA)

System LX and cross-memory services – part 2

This month we conclude the code for a batch program that displays all cross-memory connections.

```
*********************************************************************
* This routine processes one XMSE in PCAUTH's address space.        *
* Most of this routine executes in AR (Access Register) mode.       *
* R12 : used to base XMSE (AR mode)                                 *
* R11 : used to base SETC (AR mode)                                 *
* R1Ø : used as work register                                       *
* R9  : used as work register                                       *
*********************************************************************
*
PROCESS_XMSE   DS ØH
         BAKR  R14,Ø                   Push environment into stack
         BAS   R14,WRITE_LISTXME2_LINE  Header line with jobname & asid
         LAM   R12,R12,MYALET          Load the PCAUTH's ALET
         USING XMSE,R12                Establish addressability to XMSE
         L     R12,XMSECUR#            Load current XMSE addr
         USING SETC,R11                Establish addressability to SETC
         CPYA  R11,R12                 Copy the ALET into R11
*
         SAC   512                     Switch to AR mode
*
         CLC   XMSEACRO,=C'XMSE'       Good acronym ?
         BNE   BADACRO                 No, problem.
         MVC   HEX1,XMSESETC           Let's see SETC
         BAS   R14,CONVERT_TO_CHAR
         MVC   $XM1SETC,HEX2
*
         L     R11,XMSESETC
         TM    SETCFLG1,SYSTEMLX       System LX ?
```

```
         BZ    NOSYSLX                No, let see specific connections
*
         SAC   Ø                      Go back into home mode
*
         MVC   $XM1SYLX,=C'SysLX'
         MVC   $XM2LINE(LXM2LIN2),$XM2LIN2
         BAS   R14,WRITE_LISTXME2_LINE  System LX line
         BAS   R14,WRITE_LISTXME2_LINE  One blank line
         BAS   R14,WRITE_LISTXME2_LINE  Second blank line
         PR                           Pop stack and return to caller
*
NOSYSLX  DS    ØH                     Specific cross-memory connections
*
         SAC   Ø                      Go back into home mode
*
         MVC   $XM2LINE(LXM2LIN3),$XM2LIN3
         BAS   R14,WRITE_LISTXME2_LINE  Title line for this addr space
*
         SAC   512                    Switch to AR mode
*
         MVC   HEX1,SETCTO            To and from connections....
         BAS   R14,CONVERT_TO_CHAR
         MVC   $XM1TO,HEX2            To connections........
         MVC   $XM1FROM,HEX2+4        From connections......
*
         LH    R9,SETCTO              Calculate number of connections
         LH    R1Ø,SETCFROM           To + From connections
         AR    R9,R1Ø                 R9 = index for XMSELOOP
         LTR   R9,R9                  No connection ?
         BZ    XMSEEND                Strange, but possible .....
*
XMSELOOP DS    ØH
         MVC   HEX1,SETCXMSE          Put connected XMSE on output line
         BAS   R14,CONVERT_TO_CHAR
         MVC   $XM2XMSE,HEX2
         MVC   $XM2TYPE,=CL4' To'     Init field
         TM    SETCXMSE,#HIGHON       Is it a 'To' connection type ?
         BO    XMSEØØ1Ø               Yes, carry on
         MVC   $XM2TYPE,=CL4'From'    No, it's a 'From' connection type
XMSEØØ1Ø DS    ØH
         TM    SETCXMSE+3,#LOWON      Is it a valid XMSE ?
         BZ    XMSEØØ2Ø               No, don't go further
         MVC   $XM2JBN2,=Cl8'Not Used' This connection is no
         B     XMSEØØ3Ø                longer used.
XMSEØØ2Ø DS    ØH
         L     R12,SETCXMSE           R12 can be reused
         MVC   $XM2JBN2,XMSEJBNA      Get connected jobname
         MVC   HEX1,XMSEASID          Get connected asid number
         BAS   R14,CONVERT_TO_CHAR
         MVC   $XM2ASI2,HEX2
```

```
XMSEØØ3Ø DS    ØH
         SAC   Ø                      Go back into home mode
         BAS   R14,WRITE_LISTXME2_LINE
         SAC   512                    Switch to AR mode
         LA    R11,4(,R11)            Next connected XMSE
         BCT   R9,XMSELOOP            Let's rock'n roll again
*
XMSEEND  DS    ØH
*
         SAC   Ø                      Go back into home mode
*
         BAS   R14,WRITE_LISTXME2_LINE  One blank line
         BAS   R14,WRITE_LISTXME2_LINE  Second blank line
*
         PR                           Pop stack and return to caller
*
BADACRO DS    ØH
         SAC   Ø                      Go back into home mode
         WTO   'REXMEMØ5  Problem with XMSE chain into  ',ROUTCDE=11
         WTO   'REXMEMØ5  the PCAUTH EPVT.              ',ROUTCDE=11
         OI    #PGMFLAG,#BADACRO      Flag on
         PR                           Pop stack and return to caller
         DROP  R12
         DROP  R11
         EJECT
*********************************************************************
* This routine writes a line on LISTXME1 and reinits current line.   *
*********************************************************************
WRITE_LISTXME1_LINE DS ØH
         BAKR  R14,Ø           Push environment into stack
         PUT   LISTXME1,$XM1LINE
         MVI   $XM1LINE,C' '
         MVC   $XM1LINE+1(L'$XM1LINE-1),$XM1LINE
         PR                    Pop stack and return to caller
         EJECT
*********************************************************************
* This routine writes a line on LISTXME2 and reinits current line.   *
*********************************************************************
WRITE_LISTXME2_LINE DS ØH
         BAKR  R14,Ø           Push environment into stack
         PUT   LISTXME2,$XM2LINE
         MVI   $XM2LINE,C' '
         MVC   $XM2LINE+1(L'$XM2LINE-1),$XM2LINE
         PR                    Pop stack and return to caller
         EJECT
*********************************************************************
* This routine frees the ALET.                                       *
*********************************************************************
ALESERV_DEL    DS ØH
         BAKR  R14,Ø                  Push environment into stack
```

```
         ALESERV DELETE,ALET=MYALET,CHKEAX=NO
         PR                          Pop stack and return to caller
         EJECT
*********************************************************************
* This routine closes all DCBs.                                     *
*********************************************************************
CLOSDCBS DS    ØH
         BAKR  R14,Ø           Push environment into stack
         CLOSE (LISTXME1)
         CLOSE (LISTXME2)
         PR                    Pop stack and return to caller
         EJECT
*********************************************************************
* This routine translates hexadecimal into printable format.
* At entry, HEX1 contains the word to translate. It will return the
* result into the double word HEX2.
* R1Ø is used as work register.
*********************************************************************
CONVERT_TO_CHAR DS ØH
         BAKR  R14,Ø                  Push environment into stack
         XR    R1Ø,R1Ø                Wipe out register
         IC    R1Ø,HEX1               LOAD FIRST BYTE
         SRL   R1Ø,4                  ELIMINATE 4 RIGHT MOST BITS
         STC   R1Ø,HEX2               SAVE FIRST 4 BITS
         IC    R1Ø,HEX1               LOAD FIRST BYTE
         SLL   R1Ø,28                 ELIMINATE 4 LEFT MOST BITS
         SRL   R1Ø,28
         STC   R1Ø,HEX2+1             SAVE SECOND SET OF 4 BITS
         IC    R1Ø,HEX1+1             LOAD SECOND BYTE
         SRL   R1Ø,4                  ELIMINATE 4 RIGHT MOST BITS
         STC   R1Ø,HEX2+2             SAVE THIRD SET OF 4 BITS
         IC    R1Ø,HEX1+1             LOAD SECOND BYTE
         SLL   R1Ø,28                 ELIMINATE 4 LEFT MOST BITS
         SRL   R1Ø,28
         STC   R1Ø,HEX2+3             SAVE FOURTH SET OF 4 BITS
         IC    R1Ø,HEX1+2             LOAD THIRD  BYTE
         SRL   R1Ø,4                  ELIMINATE 4 RIGHT MOST BITS
         STC   R1Ø,HEX2+4             SAVE FIFTH  SET OF 4 BITS
         IC    R1Ø,HEX1+2             LOAD THIRD  BYTE
         SLL   R1Ø,28                 ELIMINATE 4 LEFT MOST BITS
         SRL   R1Ø,28
         STC   R1Ø,HEX2+5             SAVE SIXTH  SET OF 4 BITS
         IC    R1Ø,HEX1+3             LOAD FOURTH BYTE
         SRL   R1Ø,4                  ELIMINATE 4 RIGHT MOST BITS
         STC   R1Ø,HEX2+6             SAVE SEVENTH SET OF 4 BITS
         IC    R1Ø,HEX1+3             LOAD FOURTH BYTE
         SLL   R1Ø,28                 ELIMINATE 4 LEFT MOST BITS
         SRL   R1Ø,28
         STC   R1Ø,HEX2+7             SAVE EIGHTH SET OF 4 BITS
         TR    HEX2(L'HEX2),TRTAB     TRANSLATE TO PRINTABLE CHAR
```

```
         XC    HEX1,HEX1              CLEAR FIELD
         PR                           Pop stack and return to caller
         EJECT
*********************************************************************
* This routine checks RC, restores registers and returns control.
*********************************************************************
RETURN   DS    ØH
         LA    R15,24                 INIT R15
         TM    #PGMFLAG,#NOTAUTH      CHECK NOT AUTHORIZED FLAG
         BO    EXIT                   IF SET, EXIT WITH RC=24
         LA    R15,2Ø                 INIT R15
         TM    #PGMFLAG,#OPENERR      CHECK OPEN ERROR FLAG
         BO    EXIT                   IF SET, EXIT WITH RC=2Ø
         LA    R15,16                 INIT R15
         TM    #PGMFLAG,#PCANOTF      CHECK PCAUTH not found FLAG
         BO    EXIT                   IF SET, EXIT WITH RC=16
         LA    R15,12                 INIT R15
         TM    #PGMFLAG,#ALETNOK      CHECK Alet not ok FLAG
         BO    EXIT                   IF SET, EXIT WITH RC=16
         LA    R15,8                  INIT R15
         TM    #PGMFLAG,#BADACRO      CHECK Bad acronym FLAG
         BO    EXIT                   IF SET, EXIT WITH RC=8
         LA    R15,Ø                  IF NOT, EXIT WITH RC=ØØ
EXIT     RCNTL RC=(15)
         EJECT
*********************************************************************
         TITLE 'REXMEM literals.'
*********************************************************************
         LTORG
         EJECT ,
*********************************************************************
         TITLE 'REXMEM Module Workarea'
*
TRTAB    DC   X'FØF1F2F3F4F5F6F7F8F9' CHARACTERS Ø123456789
         DC   X'C1C2C3C4C5C6'                   ABCDEF
*
ASSBPC#  DS    F              To save PCAUTH's ASSB address
XMSECUR# DS    F              To save current XMSE address
MYALET   DS    F              To save target ALET
HEX1     DS    F
HEX2     DS    D
*
#PGMFLAG DC    B'00000000'    Flag used for internal logic
#NOTAUTH EQU   B'10000000'    Not authorized program
#OPENERR EQU   B'01000000'    Error opening LISTXME1
#PCANOTF EQU   B'00100000'    We didn't find PCAUTH
#ALETNOK EQU   B'00010000'    We didn't get the ALET (cross-mem)
#BADACRO EQU   B'00001000'    Problem in scanning the XMSE chain
*
#HIGHON  EQU   B'10000000'    High bit on
```

```
#LOWON   EQU   B'00000001'     Low  bit on
*********************************************************************
* Print lines definitions.                                          *
*********************************************************************
$XM1LIN2 DS  0H
         DC  C'0XmseAddr Job Name Asid XmsePrev XmseNext AscbAddr     '
         DC  C'        XmseSetc       To  From'
XMSLIN2L EQU *-$XM1LIN2
*
$XM1LINE DC  CL133' '          Output line for LISTXME1
$XM1ASA  EQU $XM1LINE,1
$XM1ASCB EQU $XM1ASA+1,8       Ascb addr
$XM1SE01 EQU $XM1ASCB+8,1
$XM1JBNA EQU $XM1SE01+1,8      Jobname
$XM1SE02 EQU $XM1JBNA+8,1
$XM1ASID EQU $XM1SE02+1,4      Asid number
$XM1SE03 EQU $XM1ASID+4,1
$XM1ASTE EQU $XM1SE03+1,8      Addr space second table
$XM1SE04 EQU $XM1ASTE+8,1
$XM1LTOV EQU $XM1SE04+1,8      Linkage table origin virtual addr
$XM1SE05 EQU $XM1LTOV+8,1
$XM1ATOV EQU $XM1SE05+1,8      Authorization table origin virtual addr
$XM1SE06 EQU $XM1ATOV+8,1
$XM1ETC  EQU $XM1SE06+1,4      Num of entry tables owned by this adrspc
$XM1SE07 EQU $XM1ETC+4,1
$XM1ETCN EQU $XM1SE07+1,4      Num of connections to entry tables
$XM1SE08 EQU $XM1ETCN+4,1
$XM1LXR  EQU $XM1SE08+1,4      Num of linkage indexes reserved
$XM1SE09 EQU $XM1LXR+4,1
$XM1AXR  EQU $XM1SE09+1,4      Num of authorization indexes reserved
$XM1SE10 EQU $XM1AXR+4,1
$XM1XMSE EQU $XM1SE10+1,8      Cross-memory services block addr
$XM1SE11 EQU $XM1XMSE+8,1
$XM1SETC EQU $XM1SE11+1,8      SETC addr
$XM1SE12 EQU $XM1SETC+8,1
$XM1SYLX EQU $XM1SE12+1,5      This LX is System LX
$XM1SE13 EQU $XM1SYLX+5,1
$XM1TO   EQU $XM1SE13+1,4      Number of 'TO' connections
$XM1SE14 EQU $XM1TO+4,1
$XM1FROM EQU $XM1SE14+1,4      Number of 'FROM' connections
$XM1SE15 EQU $XM1FROM+4,1
*
$XM2LINE DC  CL133' '          Output line for LISTXME2
$XM2ASA  EQU $XM2LINE,1
$XM2JBNA EQU $XM2ASA+1,8       Jobname
$XM2SE01 EQU $XM2JBNA+8,1
$XM2ASID EQU $XM2SE01+1,4      Asid
$XM2SE02 EQU $XM2ASID+4,1
*
$XM2TYPE EQU $XM2ASA+1,4       Type of connection (To or From)
```

```
$XM2SE1Ø EQU $XM2TYPE+4,1
$XM2XMSE EQU $XM2SE1Ø+1,8      Cross-memory services block addr
$XM2SE11 EQU $XM2XMSE+8,1
$XM2JBN2 EQU $XM2SE11+1,8      Jobname
$XM2SE12 EQU $XM2JBN2+8,1
$XM2ASI2 EQU $XM2SE12+1,4      Asid
$XM2SE13 EQU $XM2ASI2+4,1
*
$XM2LIN2 DS  ØH
         DC  C'  System LX bit is on, this address space has connectivi*
               ty with all other asids.'
LXM2LIN2 EQU *-$XM2LIN2
*
$XM2LIN3 DS  ØH
         DC  C' Type     Xmse  Jobname Asid'
LXM2LIN3 EQU *-$XM2LIN3
*********************************************************************
* THE DCBS                                                          *
*********************************************************************
         DS    ØH
LISTXME1 DCB   DDNAME=LISTXME1,MACRF=(PM),RECFM=FBA,LRECL=133,         *
               DSORG=PS
LISTXME2 DCB   DDNAME=LISTXME2,MACRF=(PM),RECFM=FBA,LRECL=133,         *
               DSORG=PS
         EJECT
*********************************************************************
* Dsects                                                            *
*********************************************************************
XMSE     DSECT
XMSEACRO DS    CL4               Acronym 'XMSE'
XMSESETC DS    F                 SETC address
         DS    CL8
XMSEPREV DS    F                 Previous XMSE block
XMSENEXT DS    F                 Next XMSE block
         DS    CL4
XMSEJBNA DS    CL8               Job Name that own this XMSE
XMSEASID DS    CL2               Asid hex that own this XMSE
*
SETC     DSECT
SETCACRO DS    CL4               Acronym 'SETC'
         DS    CL2
SETCFLG1 DS    X                 Flags
SYSTEMLX EQU   B'10000000'       Flag for System LX
         DS    X
         DS    CL12
SETCTO   DS    H                 Number of 'TO'   connections
SETCFROM DS    H                 Number of 'FROM' connections
         DS    CL8
SETCXMSE DS    F                 First connected XMSE
*
```

```
CVT      DSECT=YES,LIST=YES
EJECT
IHAASVT  LIST=YES
EJECT
IHAASCB  LIST=YES
EJECT
IHAASSB  LIST=YES
EJECT
DCBD     DEVD=(DA,TA),DSORG=(QS,BS)
END
```

Michel Joly
Systems Programmer (France)

Why not share your expertise and earn money at the same time? *MVS Update* is looking for program code, CLISTs, REXX EXECs, JavaScript, etc that experienced users of MVS (OS/390, z/OS) have written to make their life, or the lives of their users, easier. We are also looking for explanatory articles, and hints and tips, from experienced users. We would also like suggestions on how to improve MVS performance.

We will publish your article (after vetting by our expert panel) and send you a cheque, as payment, and two copies of the issue containing the article once it has been published. Articles can be of any length and should be e-mailed to the editor, Trevor Eddolls, at trevore@xephon.com.

A free copy of our *Notes for Contributors*, which includes information about payment rates, is available from our Web site at www.xephon.com/nfc.

Multi-tasking with IBM C/C++ programs

Inter- and intra-address space multi-tasking is a very powerful capability providing one of the very basic tools associated with MVS and all its successor operating systems. Multiple code paths can be active simultaneously either within the same address space or in secondary address spaces, allowing for very complex application support. This capability extends to programs written in IBM C/C++ and a number of techniques are inherently available for this purpose.

INHERENT MULTI-TASKING TECHNIQUES

Several techniques are available for employing multi-tasking functionality in IBM C/C++. Let's discuss the more common techniques – fork(), spawn(), pthread_create(), and the C Multi-Tasking Facility (MTF).

THE FORK() FUNCTION

One of the classic multi-tasking tools in C/C++ is the fork() function call. The fork() function performs as follows.

The issuer of the fork() function (the parent process) creates a new unit of work (the child process) that, for all intents and purposes, is a clone of the parent. Execution in both the parent and the child process continues from the point immediately following the fork() function call. To determine whether this is a parent or a child process, a simple check of the return code from fork() can be made. If the return code is less than 0, the fork() function call has failed and the errno variable can be examined to obtain additional information regarding the failure. If the return code is greater than 0, program execution is in the parent process and the return code value from the fork() function call is the process id (PID) of the created child process. If the return code is equal to 0, program execution is in the child process. Programmatically, this could look

something like:

```
pid = fork();
if (pid < 0)
{
  printf("fork() failed - errno=%d\n",errno);
  return(-1);
}
else if (pid > 0)
{
  printf("Executing in the parent process.  Child PID is %d\n",pid);
}
else
{
  printf("Executing in the child process.\n");
}
```

The fork() function call has the following drawbacks:

- It can be used only in programs running in problem state, key 8.
- The child process runs in a separate address space from that of the parent process.
- The parent process task issuing the fork() is copied into the child address space.
- MVS files opened in the parent process are not opened in the child process with the exception of TASKLIB, STEPLIB, and/or JOBLIB. The child process maintains the same MVS program search order as the initiating parent process, but does not maintain access to other MVS datasets allocated to the parent process.

THE SPAWN() FUNCTION

A second multi-tasking method is the spawn() function. Here are some of its functional highlights:

- The child process will inherit the parent process's TASKLIB, STEPLIB, and JOBLIB allocations unless a STEPLIB environment variable is used for the spawned process.
- The setting of the _BPX_SHAREAS environment variable

indicates whether the child process should run as a subtask within the parent process address space or whether the child process should run in its own address space. The following _BPX_SHAREAS options are supported:

- NO – the child process is created in a separate address space. This is the default.
- REUSE – the child process is created as a subtask in an existing task structure unless conditions exist that force the child process to initiate a new task structure.
- MUST – the child process must run in the same address space as the parent or the spawn request will fail. Possible reasons for failure include:
  - if the set UID or set GID of the spawned program differs from the effective UID or the effective GID of the parent
  - the program to be run is APF authorized but the parent is not
  - the program to be run is unauthorized but the parent program is APF authorized
  - the parent process address space does not have sufficient resources.
- YES – the child process will run as a subtask in the same address space as the parent unless conditions exist that force the creation of a child process address space.

• Spawn() initiates the requested program from its entry point address. In contrast, with fork() the child process program begins execution from the instruction following the fork() instruction (not the entry point of the program) and is always a copy of the parent process program. Spawn() can be used to execute a completely different program from that of the caller.

The spawn() function has the drawback that the requested spawn program must reside in the USS HFS.

THE PTHREAD_CREATE() FUNCTION

A third multi-tasking option supported by C/C++ is the pthread_create() function. The pthread_create() function works as follows:

- Subtasks are created for each successfully initiated pthread_create() request.
- Parameters can be passed to the target program through a pthread_create() parameter.
- Pthread_create() can be used from either an authorized or unauthorized environment.

Potential drawbacks of the pthread_create() function include:

- Subtask routines initiated by pthread_create() must exist in the load module of the caller, although the routines do run under a separate driver subtask program (BPXPTATT in module BPXINLPA).
- The environment must be POSIX(ON).

There is a suite of pthread_-related function calls. Some of the more interesting and relevant ones are:

- Pthread_detach() – cleans up resources used for a pthread_create().
- Pthread_exit() – terminates the subtask initiated by a pthread_create().
- Pthread_join() – can be used to wait for a pthread_create() initiated subtask to complete.

THE IBM C/C++ MULTI-TASKING FACILITY (MTF)

A fourth multi-tasking option is the C/C++ Multi Tasking Facility. MTF works as follows:

- A tinit() function call is used to define the parallel load module that will be used for multi-tasking requests as well as the maximum number of concurrent requests that can be active.
- A tsched() function call is made (to a CSECT that exists in the tinit() requested load module) to activate an MTF subtask.
- A tsyncro() function call is used to clean up resources and wait for the completion of any tsched() initiated requests.
- A tterm() function call terminates the currently defined MTF environment.

MTF has the following drawbacks:

- The number of possible concurrent requests is restricted by the tinit() set-up function.
- All possible multi-tasking routines must be included in the tinit() parallel function load module.
- If an abend occurs in any tsched() request, the active MTF environment becomes ineligible for use and must be terminated and reinitialized before further MTF activity can occur.
- It requires a POSIX(OFF) environment.

ANOTHER OPTION

For anyone familiar with the ATTACH Assembler macro, the multi-tasking technique just discussed that most closely resembles ATTACH is the pthread_create() function. The drawbacks of the pthread_create() function are just sufficient to warrant the creation of a new function pair, ATTACH() and DETACH(). The ATTACH() function is capable of attaching programs that are available in an address space's program search order. The ATTACH() function requires a minimum of five parameters. All parameters in excess of the minimum are

presumed to be parameters that are to be passed to the attached program. The five required parameters are:

- Address of the 8-character program name (right padded with blanks) of the program to be attached.
- Address of an ATTACH() function workarea provided by the initiating program. The workarea must be a minimum of 256 bytes plus an additional four bytes for each optional parameter that will be passed to the attached program. For example, if nine parameters are passed in the ATTACH() function call, the first five are assumed to be the default required parameters. The remaining four parameters will be passed through to the attached program. In this case, the ATTACH() function workarea address must point to a workarea that is at least 272 bytes (the minimum 256 bytes plus 16 bytes (4*4) for the four parameters to be passed to the attached program). This workarea should not be used, modified, or freed by the program calling the ATTACH() function until the attached program has completed execution. If it is, the result will be unpredictable.
- Address of an ECB area. This parameter can be a null pointer if completion of the attached program will not be monitored.
- Address of a TCB address return area. This parameter can be a null pointer if the TCB address is not required by the issuer of the ATTACH() function.
- Address of a TASKLIB DCB area address. This parameter can be a null pointer if the address space's existing search order will be used to locate the ATTACH() specified program.

If an ECB area address has been provided in the ATTACH() function call, a DETACH() function call will have to be made for the corresponding TCB. This will allow for clean-up processing and prevent A03 abends from occurring during address space termination. The DETACH() function requires two parameters:

- Address of the TCB area address.
- Address of the STAE option indicator. The STAE option indicator should be either STAE or NOSTAE. For specific details on the STAE option indicator, check out the DETACH macro parameter descriptions in the z/OS *MVS Assembler Services Reference* manual.

Comments in the program source for ATTACH and DETACH provide more details on the function arguments.

PROGRAM COMPILATION, LINKAGE, AND EXECUTION

Included with this article are four programs. These are:

- The ATTACH() function Assembler program.
- The DETACH() function Assembler program.
- The TESTATT C program, which provides example usage of ATTACH() and DETACH().
- The ATTPGM1 Assembler program, which is attached by the TESTATT example program.

ATTACH, DETACH, and ATTPGM1 can be assembled with a standard assembly job. Datasets SYS1.MACLIB, SYS1.MODGEN, and CEE.SCEEMAC should be included in the assembly job's SYSLIB DD concatenation.

The TESTATT C program should be compiled and prelinked as per normal procedures for these operations – be sure to specify the RENT option on the compile. Also, be sure to change all occurrences of [to X'AD' and all occurrences of] to X'BD' prior to running the compile.

Below is a sample job that linkedits the ATTPGM1 and TESTATT programs to create executable modules:

```
//IEWL     EXEC  PGM=HEWLHØ96,PARM='XREF,LIST,MAP,RENT'
//SYSPRINT DD    SYSOUT=*
//OBJECT   DD    DSN=object.code.pds,DISP=SHR
//SYSLIB   DD    DSN=CEE.SCEELKED,DISP=SHR
//SYSLMOD  DD    DSN=load.library,DISP=SHR
```

```
//SYSLIN   DD    *
   INCLUDE OBJECT(TESTATT)    OBJ MODULE FOR TESTATT AFTER PRELINK
   INCLUDE OBJECT(ATTACH)     OBJ MODULE FOR ATTACH FUNCTION
   INCLUDE OBJECT(DETACH)     OBJ MODULE FOR DETACH FUNCTION
   ENTRY   CEESTART
   NAME    TESTATT(R)
   INCLUDE OBJECT(ATTPGM1)    OBJ MODULE FOR ATTPGM1 PROGRAM
   NAME    ATTPGM1(R)
```

Here is sample JCL for running a test:

```
//TESTATT  EXEC  PGM=TESTATT
//STEPLIB  DD    DSN=load.library,DISP=SHR
//SYSPRINT DD    SYSOUT=*
```

CONCLUSION

Using the ATTACH()/DETACH() function pair in a C or C++ program environment provides much more control over a multi-tasking environment than any of the inherent methods provided for IBM C/C++ programs. If you have the need for multi-tasking C/C++ applications, I'm confident that you will see the benefit of using these functions.

ATTACH ASM

```
*-------------------------------------------------------------------*
*   This program provides support for a C/C++ ATTACH() function.    *
*   It is designed to function similarly to the ATTACH macro for    *
*   Assembler programs.  The ATTACH() function supports a basic     *
*   program ATTACH with parameter passing support.  This version    *
*   of the function does not support the more esoteric ATTACH macro *
*   parameters, but does provide for specifying the address of a DCB *
*   for an open TASKLIB DD.  As well, for tasks that need to be     *
*   waited on for completion, the ATTACH() function supports the    *
*   passing of an ECB area address and a return area for the TCB    *
*   address.                                                        *
*                                                                   *
*   For this program the following register usage is in effect:     *
*                                                                   *
*   RØ - R1    :  work registers, but generally available for use   *
*                 by calls to system functions                      *
*   R2         :  used to save the incoming parameter address       *
*   R3 - R7    :  work registers                                    *
*   R8         :  used as base register for the required incoming   *
*                 workarea                                          *
```

```
*   R9        :  work register                                          *
*   R1Ø - R11 :  reserved (future base register expansion)              *
*   R12       :  base register                                          *
*   R13       :  DSA/workarea address                                   *
*   R14 - R15 :  work registers, return address and return code, but    *
*                generally available for use by calls to system         *
*                functions                                              *
*--------------------------------------------------------------------*
*   Routine:     ATTACH                                                *
*                                                                      *
*   Function:    To provide MVS ATTACH capabilities from an IBM        *
*                C/C++ program.                                        *
*                                                                      *
*   Arguments:   ATTACH program name address (right pad with blanks)   *
*                ATTACH() function workarea address.  This workarea    *
*                   must be a minimum of 256 bytes plus four bytes     *
*                   for each optional parm that is passed.  It         *
*                   should not be modified by the calling program or   *
*                   used for any other ATTACH() calls while this       *
*                   task is active.                                    *
*                ECB area address (or NULL)                            *
*                TCB area address (or NULL)                            *
*                TASKLIB DCB area address (or NULL)                    *
*                Optional parms to be passed to the attached program.  *
*                   The last parm address will have the X'8Ø' flag     *
*                   set.  You can pass up to 256 optional parms.       *
*                                                                      *
*   Return:      Ø if the ATTACH is successful                         *
*               -7 ATTACH failed.  If an ECB area address has been     *
*                  provided, the ECB area contains the ATTACH          *
*                  return code.                                        *
*               -8 incorrect minimum number of parms.  The ATTACH()    *
*                  function call requires a minimum of five parms.     *
*               -9 no parms were detected on entry to ATTACH()         *
*                                                                      *
*   C usage:    i = ATTACH(&pgm_name, &attach_workarea_addr,           *
*                          &ecb, &tcb, &tasklib_dcb,                   *
*                          &opt_parm1, &opt_parm2, ... , &opt_parmn); *
*--------------------------------------------------------------------*
ATTACH   CSECT
ATTACH   AMODE 31
ATTACH   RMODE ANY
         EDCPRLG BASEREG=R12,DSALEN=WORKLEN
         USING ATTAWORK,R13            Addressability to temp storage
*--------------------------------------------------------------------*
         LTR   R1,R1                   Parms ok?
         BZ    RETNEGØ9                No - return -9
         LR    R9,R1                   Copy parm address
         L     R2,Ø(,R9)               Get buffer address
         N     R2,=X'8ØØØØØØØ'         Turn off address value
```

```
         C     R2,=X'80000000'          Is this the last parm?
         BE    RETNEG08                 Yes - return -8
         L     R2,4(,R9)                Get buffer address
         N     R2,=X'80000000'          Turn off address value
         C     R2,=X'80000000'          Is this the last parm?
         BO    RETNEG08                 Yes - return -8
         L     R2,4(,R9)                Get buffer address
         L     R8,0(,R2)                Get WORKAREA address
         USING WORKAREA,R8              Set WORKAREA addressability
         LA    R6,PARMS                 Get parm address area address
*-------------------------------------------------------------------*
*   R1 contains the address of the the incoming parms.  Check to    *
*   make sure that a valid minimum number of parameters have been   *
*   passed.                                                         *
*-------------------------------------------------------------------*
         ST    R1,PARM0                 Save incoming parm address
         LTR   R1,R1                    Parms ok?
         BZ    RETNEG09                 No - return -9
         LR    R9,R1                    Copy parm address
         L     R2,0(,R9)                Get buffer address
         ST    R2,ATTAPGM               Save pgm name address
         TM    ATTAPGM,X'80'            Is this the last parm?
         BO    RETNEG08                 Yes - return -8
         L     R2,4(,R9)                Get buffer address
         ST    R2,ATTAWRK               Save work area address
         TM    ATTAWRK,X'80'            Is this the last parm?
         BO    RETNEG08                 Yes - return -8
         L     R2,8(,R9)                Get buffer address
         ST    R2,ATTAECB               Save ECB address
         TM    ATTAECB,X'80'            Is this the last parm?
         BO    RETNEG08                 Yes - return -8
         L     R2,12(,R9)               Get buffer address
         ST    R2,ATTATCB               Save TCB address
         TM    ATTATCB,X'80'            Is this the last parm?
         BO    RETNEG08                 Yes - return -8
         L     R2,16(,R9)               Get buffer address
         ST    R2,ATTATSKL              Save TASKLIB DCB address
         LA    R14,256                  Set parm base number
         LA    R15,256                  Set parm base number
         TM    ATTATSKL,X'80'           Is this the last parm?
         BNO   MOREPRMS                 No - capture additional parms
         OI    PARMS,X'80'              Set last parm flag
         B     PASTPRMS                 Bypass parm capture
MOREPRMS DS    0H
         NI    ATTATSKL,X'7F'           Turn off the x'80' flag
         OI    FLAG1,PPARMS             Set parm flag
         LA    R1,PARMS                 Get parm addr save area addr
         LA    R15,256                  Set loop count
         LA    R9,20(,R9)               Point to first parm addr
PARMLP   DS    0H
```

```
         MVC   Ø(4,R1),Ø(R9)           Copy parm address
         TM    Ø(R1),X'8Ø'             Last parm?
         BO    PASTPRMS                Yes - we're done
         LA    R1,4(,R1)               Point to next target area
         LA    R9,4(,R9)               Point to next source area
         BCT   R15,PARMLP              Check for more
         OI    PARMS+255*4,X'8Ø'       Set last parm flag
PASTPRMS DS    ØH
         SR    R14,R15                 Calculate number of parms
         LTR   R14,R14                 Any parms?
         BZ    NOPRMS                  No - bypass initialization
         SLL   R14,2                   Multiply by 4
         LR    R7,R14                  Copy length
         LA    R6,PARMS                Get parm address area address
         LR    R14,R6                  Copy the address
         XR    R15,R15                 Set fill byte
         MVCL  R6,R14                  Clear the area
NOPRMS   DS    ØH
         L     R3,ATTAPGM              Get pgm name address
         L     R4,ATTAECB              Get ECB area address
         L     R5,ATTATCB              Get TCB area address
         L     R7,ATTATSKL             Get TASKLIB DCB area address
         LTR   R4,R4                   An ECB address?
         BZ    NODETACH                No - DETACH isn't required
         LTR   R5,R5                   A TCB address?
         BZ    NODETACH                No - DETACH isn't required
         XC    Ø(4,R4),Ø(R4)           Clear the ECB
*********************************************************************
*   ATTACH the requested program.                                   *
*********************************************************************
         MVC   ATTACHWK(ATTACHLN),ATTACHLS Copy the model
         LTR   R7,R7                   A TASKLIB DCB?
         BNZ   TASKLIB1                Yes - issue ATTACH with TASKLIB
         ATTACHX EPLOC=(R3),         ** SPECIFIED PROGRAM           **X
               ECB=(R4),             ** TARGET ECB                  **X
               MF=(E,PARMS),         ** PARM LIST ADDRESS           **X
               VL=1,                 ** SET X'8Ø' BIT ON LAST PARM  **X
               SF=(E,ATTACHWK)       ** INDICATE EXECUTE FORM       **
         LTR   R15,R15                 All's well?
         BNZ   RETNEGØ7                No - save RC in ECB area
         ST    R1,Ø(,R5)               Save TCB address
         B     RETURNOK                Return
TASKLIB1 DS    ØH
         L     R7,Ø(,R7)               Get TASKLIB DCB address
         ATTACHX EPLOC=(R3),         ** SPECIFIED PROGRAM           **X
               ECB=(R4),             ** TARGET ECB                  **X
               TASKLIB=(R7),         ** TASKLIB DCB                 **X
               MF=(E,PARMS),         ** PARM LIST ADDRESS           **X
               VL=1,                 ** SET X'8Ø' BIT ON LAST PARM  **X
               SF=(E,ATTACHWK)       ** INDICATE EXECUTE FORM       **
```

```
         LTR   R15,R15                  All's well?
         BNZ   RETNEGØ7                 No - save RC in ECB area
         ST    R1,Ø(,R5)                Save TCB address
         B     RETURNOK                 Return
NODETACH DS    ØH
*************************************************************************
*   ATTACH the requested program.                                       *
*************************************************************************
         MVC   ATTACHWK(ATTACHLN),ATTACHLS Copy the model
         LTR   R7,R7                    A TASKLIB DCB?
         BNZ   TASKLIB2                 YES - ISSUE ATTACH WITH TASKLIB
         ATTACHX EPLOC=(R3),         ** SPECIFIED PROGRAM            **X
               MF=(E,PARMS),         ** PARM LIST ADDRESS            **X
               VL=1,                 ** SET X'8Ø' BIT ON LAST PARM   **X
               SF=(E,ATTACHWK)       ** INDICATE EXECUTE FORM        **
         LTR   R15,R15                  All's well?
         BNZ   RETNEGØ7                 No - save RC in ECB area
         B     RETURNOK                 Return
TASKLIB2 DS    ØH
         L     R7,Ø(,R7)                Get TASKLIB DCB address
         ATTACHX EPLOC=(R3),         ** SPECIFIED PROGRAM            **X
               TASKLIB=(R7),         ** TASKLIB DCB                  **X
               MF=(E,PARMS),         ** PARM LIST ADDRESS            **X
               VL=1,                 ** SET X'8Ø' BIT ON LAST PARM   **X
               SF=(E,ATTACHWK)       ** INDICATE EXECUTE FORM        **
         LTR   R15,R15                  All's well?
         BNZ   RETNEGØ7                 No - save RC in ECB area
         B     RETURNOK                 Return
*************************************************************************
RETURNOK EQU   *
*************************************************************************
         MVC   RETCODE(4),=F'Ø'         Set return code to Ø
*************************************************************************
         B     RETURN                   Return
RETNEGØ7 EQU   *
         ST    R15,Ø(,R4)               Save RC in ECB area
         MVC   RETCODE(4),=F'-7'        Set return code to -7
         B     RETURN                   Return
RETNEGØ8 EQU   *
         MVC   RETCODE(4),=F'-8'        Set return code to -8
         B     RETURN                   Return
RETNEGØ9 EQU   *
         MVC   RETCODE(4),=F'-9'        Set return code to -9
         B     RETURN                   Return
RETURN   EQU   *
         L     R15,RETCODE              Load return code
         EDCEPIL
*************************************************************************
ATTACHLS ATTACHX SF=L
ATTACHLN EQU   *-ATTACHLS
```

```
*********************************************************************
         LTORG
*********************************************************************
ATTAWORK EDCDSAD
WORKLEN  EQU   *-ATTAWORK
WORKAREA DSECT
ATTAWRKL DS    F                       Length of this WORKAREA
PARMØ    DS    F                       Address of incoming parms
ATTAPGM  DS    F                       Address of ATTACH pgm name
ATTAWRK  DS    F                       Address of this WORKAREA
ATTAECB  DS    F                       Address of the ATTACH ECB
ATTATCB  DS    F                       Address of TCB addr return area
ATTATSKL DS    F                       Address of TASKLIB DCB addr
RETCODE  DS    F                       Return code
DBL1     DS    2D                      Dbl work work area
DBL2     DS    2D                      Dbl work work area
FLAGS    DS    ØF
FLAG1    DS    XL1
PPARMS   EQU   X'8Ø'
FLAG2    DS    XL1
FLAG3    DS    XL1
FLAG4    DS    XL1
ATTACHWK DS    ØD,CL(ATTACHLN)
PARMS    DS    256F                    Incoming parm addresses
WORKLEN2 EQU   *-PARMØ
RØ       EQU   Ø
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R1Ø      EQU   1Ø
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         END
```

DETACH ASM

```
*--------------------------------------------------------------------*
*   This file contains the Assembler support code for a DETACH()     *
*   function call.  This routine is intended to be called from       *
*   IBM C/C++ programs and is used to remove a previously created    *
```

```
*   subtask.                                                           *
*                                                                      *
*   Register Usage Conventions:                                        *
*                                                                      *
*   RØ - R1   :  work registers, but generally available for use       *
*                by calls to system functions                          *
*   R2        :  used to save the incoming parameter address           *
*   R3 - R9   :  work registers                                        *
*   R1Ø - R11 :  reserved (future base register expansion)             *
*   R12       :  base register                                         *
*   R13       :  DSA/workarea address                                  *
*   R14 - R15 :  work registers, return address and return code, but   *
*                generally available for use by calls to system        *
*                functions                                             *
*--------------------------------------------------------------------*
*--------------------------------------------------------------------*
*   Routine:     DETACH                                                *
*                                                                      *
*   Function:    To provide MVS DETACH capabilities from an IBM        *
*                C/C++ program.                                        *
*                                                                      *
*   Arguments:   TCB area address                                      *
*                STAE option indicator address (STAE/NOSTAE)           *
*                                                                      *
*   Return:      Ø if the DETACH is successful                         *
*               -1 task was DETACHed while active                      *
*               -8 incorrect number of parms.  the DETACH() function   *
*                  call requires a tcb address parm.                   *
*               -9 no parms were detected on entry to DETACH()         *
*                                                                      *
*   C Usage:    i = DETACH(&tcb, &stae_opt);                           *
*--------------------------------------------------------------------*
DETACH   CSECT
DETACH   AMODE 31
DETACH   RMODE ANY
         EDCPRLG BASEREG=R12,DSALEN=WORKLEN
         LR    R2,R1                    Save incoming parm addr
         USING DETAWORK,R13             Addressability to temp storage
*--------------------------------------------------------------------*
         ST    R2,PARMØ                 Save incoming parm address
         LTR   R2,R2                    Parms ok?
         BZ    RETNEGØ9                 No - return -9
         L     R9,Ø(,R2)                Get buffer address
         ST    R9,DETATCB               Save TCB address
         TM    DETATCB,X'8Ø'            Is this the last parm?
         BO    RETNEGØ8                 Yes - return -8
         L     R9,4(,R2)                Get buffer address
         ST    R9,DETASTAE              Save STAE option indicator addr
         TM    DETASTAE,X'8Ø'           Is this the last parm?
         BNO   RETNEGØ8                 No - return -8
```

```
         L     R1,DETASTAE              Get STAE option indicator addr
         CLC   Ø(4,R1),=C'STAE'         STAE=YES?
         BE    STAEYES                  Yes - DETACH with STAE=YES
         B     STAENO                   No - DETACH with STAE=NO
*-------------------------------------------------------------------*
*   DETACH the requested TCB (STAE=YES).                            *
*-------------------------------------------------------------------*
STAEYES  DS    ØH
         L     R5,DETATCB               Get TCB area addr
         DETACH (R5),STAE=YES           DETACH
         LTR   R15,R15                  All's well?
         BNZ   RETNEGØ1                 No - return -1
         MVC   RETCODE(4),=F'Ø'         Set return code
*-------------------------------------------------------------------*
         B     RETURN                   Return
*-------------------------------------------------------------------*
*                                                                   *
*   DETACH the requested TCB (STAE=NO).                             *
*                                                                   *
*-------------------------------------------------------------------*
STAENO   DS    ØH
         L     R5,DETATCB               Get TCB area addr
         DETACH (R5),STAE=NO            DETACH
         LTR   R15,R15                  All's well?
         BNZ   RETNEGØ1                 No - return -1
         MVC   RETCODE(4),=F'Ø'         Set return code
*-------------------------------------------------------------------*
         B     RETURN                   Return
RETNEGØ1 DS    ØH
         MVC   RETCODE(4),=F'-1'        Set return code to -1
         B     RETURN                   Return
RETNEGØ8 DS    ØH
         MVC   RETCODE(4),=F'-8'        Set return code to -8
         B     RETURN                   Return
RETNEGØ9 DS    ØH
         MVC   RETCODE(4),=F'-9'        Set return code to -9
         B     RETURN                   Return
RETURN   DS    ØH
         L     R5,RETCODE               Copy the return code
         LR    R15,R5                   Load return code
         EDCEPIL
*-------------------------------------------------------------------*
         LTORG
*-------------------------------------------------------------------*
DETAWORK EDCDSAD
PARMØ    DS    F                        Address of incoming parms
DETATCB  DS    F                        Address of TCB
DETASTAE DS    F                        STAE option indicator address
RETCODE  DS    F                        Return code
DBL1     DS    2D                       Dbl word work area
```

```
DBL2     DS    2D                        Dbl word work area
WORKLEN  EQU   *-DETAWORK
RØ       EQU   Ø
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R1Ø      EQU   1Ø
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         END
```

TESTATT C

```
/*
 *  The TESTATT program is designed to test the ATTACH()/DETACH()
 *  function pair.  The ATTACH() function can be used as an alternative
 *  to other IBM C/C++ multi-tasking tools such as fork(), spawn(),
 *  pthread_create(), or the C Multi-Tasking Facility (MTF).
 *
 *  This program can be compiled as either an IBM C or an IBM C++
 *  program.  Use standard C or C++ compile and prelink procedures.
 *
 *  Be sure to convert all occurrences of '[' to x'AD' and all
 *  occurrences of ']' to x'BD' prior to performing the program
 *  compile.
 */
#pragma runopts("POSIX(ON)")
#define _POSIX_SOURCE
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
/*  Indicate to the compiler that standard OS linkage will be used.  */
#ifdef __cplusplus
  extern "OS" int ATTACH(char*, unsigned char**, unsigned int*,
                         unsigned int*, unsigned int*, ...);
  extern "OS" int DETACH(unsigned int*, char*);
#else
  #pragma linkage (ATTACH, OS)
```

```
  #pragma linkage (DETACH, OS)
#endif
main(int argc, char *argv[])
{
  int i,j,k;
  unsigned char *attach_work1;
  unsigned int ecb1;
  unsigned int tcb1;
  unsigned int c1, d1, e1, f1, i1;
  unsigned char *attach_work2;
  unsigned int ecb2;
  unsigned int tcb2;
  unsigned int c2, d2, e2, f2, i2;
  unsigned char *attach_work3;
  unsigned int ecb3;
  unsigned int tcb3;
  unsigned int c3, d3, e3, f3, i3;
/*  Acquire working storage for first subtask.  */
  attach_work1 = (unsigned char *)calloc(2048,1);
  if (attach_work1 == NULL)
  {
   printf("Unable to obtain working storage.\n");
   return(-1);
  }
  memset(attach_work1, 0, 2048);
  attach_work1[2] = 0x08;
  attach_work1[3] = 0x00;
/*  Acquire working storage for second subtask.  */
  attach_work2 = (unsigned char *)calloc(2048,1);
  if (attach_work2 == NULL)
  {
   printf("Unable to obtain working storage.\n");
   return(-1);
  }
  memset(attach_work2, 0, 2048);
  attach_work2[2] = 0x08;
  attach_work2[3] = 0x00;
/*  Acquire working storage for third subtask.  */
  attach_work3 = (unsigned char *)calloc(2048,1);
  if (attach_work3 == NULL)
  {
   printf("Unable to obtain working storage.\n");
   return(-1);
  }
  memset(attach_work3, 0, 2048);
  attach_work3[2] = 0x08;
  attach_work3[3] = 0x00;
  j = 240;
  k = 100;
/*  ATTACH the test program three times with various different
```

```
    parameter values for verification purposes.  The ATTACH program
    name (in this case, ATTPGM1) must be right padded with blanks
    if the program name is less than 8 characters.  */
  i1 = ATTACH("ATTPGM1 ",&attach_work1,
              &ecb1,&tcb1,NULL,&j,250,"abcdef");
  i2 = ATTACH("ATTPGM1 ",&attach_work2,
              &ecb2,&tcb2,NULL,&k,1024,"123456");
  i3 = ATTACH("ATTPGM1 ",&attach_work3,
              &ecb3,&tcb3,NULL,4096,k,"987654");
/*  Isolate the task complete indicator bit in the ECB and wait for
    each subtask to complete.  */
  c1 = ecb1;
  d1 = c1 >> 24;
  e1 = d1 & 0x00000040;
  c2 = ecb2;
  d2 = c2 >> 24;
  e2 = d2 & 0x00000040;
  c3 = ecb3;
  d3 = c3 >> 24;
  e3 = d3 & 0x00000040;
  while (e1 == 0 ]] e2 == 0 ]] e3 == 0)
  {
    sleep(1);
    c1 = ecb1;
    d1 = c1 >> 24;
    e1 = d1 & 0x00000040;
    c2 = ecb2;
    d2 = c2 >> 24;
    e2 = d2 & 0x00000040;
    c3 = ecb3;
    d3 = c3 >> 24;
    e3 = d3 & 0x00000040;
  }
/*  The subtasks have all completed.  DETACH and terminate the parent
    program.  */
  i1 = DETACH(&tcb1,"STAE");
  i2 = DETACH(&tcb2,"STAE");
  i3 = DETACH(&tcb3,"STAE");
  return(0);
}
```

ATTPGM1 ASM

```
*********************************************************************
*   This program is a test program used to test out the ATTACH()    *
*   function.  The sample driver C program TESTATT is used to       *
*   ATTACH this test program any number of times.  This program     *
*   expects an incoming parameter that contains the address of the  *
*   incoming parameter list.  Three parameters are passed to this   *
```

```
*   program as follows:                                                 *
*   - address of a parameter field containing a 32-bit unsigned         *
*     integer                                                           *
*   - address of a parameter field containing a 32-bit unsigned         *
*     integer                                                           *
*   - address of a parameter field containing a null delimited          *
*     character string                                                  *
*                                                                       *
*   This program returns a return code of Ø if the incoming             *
*   parameters have been processed.  A return code of 4 is used         *
*   if no parameter data is detected.                                   *
*********************************************************************
ATTPGM1  CSECT
ATTPGM1  AMODE 31
ATTPGM1  RMODE ANY
         BAKR  R14,Ø               Stack the register values
         LR    R12,R15             Copy module base address
         USING ATTPGM1,R12         Set addressability
         LR    R11,R1              Save parm address
         LR    R3,R13              Copy savearea address
         STORAGE OBTAIN,LENGTH=WORKSIZE,LOC=ANY
         LR    RØ,R1               Copy the storage address
         LR    R13,R1              Again
         LR    R14,R1              Again
         L     R1,=A(WORKSIZE)     Get storage length
         XR    R15,R15             Set fill byte
         MVCL  RØ,R14              Sanitize the storage
*********************************************************************
         USING WORKAREA,R13        Set addressability to temp storage
         MVC   SAVEAREA+4(4),=C'F1SA'
*********************************************************************
         ST    R11,PARMADDR        Save incoming parm address
         LR    R1,R11              Copy parm address
         LTR   R1,R1               Any parameter?
         BZ    RETURNØ4            No - set return code and exit
*********************************************************************
         L     R1,16               Get CVT address
         L     R2,Ø(,R1)           Point to TCB/ASCB
         L     R3,4(,R2)           Get active TCB address
         ST    R3,DBL2             Save TCB address
         UNPK  DBL1(9),DBL2(5)     Unpack it
         NC    DBL1(8),=8X'ØF'     Turn off high order nibble
         TR    DBL1(8),=C'Ø123456789ABCDEF' Make it readable
         MVC   TCBADDR(8),DBL1     Save for later
*********************************************************************
         L     R1,PARMADDR         Load incoming parm address
         L     R3,Ø(,R1)           Get parameter address
         ICM   R5,B'1111',Ø(R3)    Copy parameter value
         CVD   R5,DBL1             Convert to decimal
         L     R15,DBL1+4          Load significant portion
```

```
         SRL   R15,4               Dump the 'sign'
         ST    R15,DBL2            Save the length
         UNPK  DBL1(9),DBL2(5)     Unpack the value
         NC    DBL1(8),=8X'ØF'     Clear high order nibbles
         TR    DBL1(8),=C'Ø123456789' Make the value readable
         MVC   WTO1WRK(WTO1LN),WTO1LST Copy WTO model
         MVC   WTO1WRK+4+25(8),DBL1  Copy readable parm length
         MVC   WTO1WRK+4+19(2),=C'#1' Set parm # indicator
         MVC   WTO1WRK+4+42(8),TCBADDR Copy TCB address
         WTO   MF=(E,WTO1WRK)      Issue WTO
*********************************************************************
         L     R1,PARMADDR         Load incoming parm address
         L     R3,4(,R1)           Get parameter address
         ICM   R5,B'1111',Ø(R3)    Copy parameter value
         CVD   R5,DBL1             Convert to decimal
         L     R15,DBL1+4          Load significant portion
         SRL   R15,4               Dump the 'sign'
         ST    R15,DBL2            Save the length
         UNPK  DBL1(9),DBL2(5)     Unpack the value
         NC    DBL1(8),=8X'ØF'     Clear high order nibbles
         TR    DBL1(8),=C'Ø123456789' Make the value readable
         MVC   WTO1WRK(WTO1LN),WTO1LST Copy WTO model
         MVC   WTO1WRK+4+25(8),DBL1  Copy readable parm length
         MVC   WTO1WRK+4+19(2),=C'#2' Set parm # indicator
         MVC   WTO1WRK+4+42(8),TCBADDR Copy TCB address
         WTO   MF=(E,WTO1WRK)      Issue WTO
*********************************************************************
         L     R1,PARMADDR         Load incoming parm address
         L     R3,8(,R1)           Get parameter address
         MVC   WTO1WRK(WTO1LN),WTO1LST Copy WTO model
         MVC   WTO1WRK+4+25(8),=8C' ' Clear the data area
         MVC   WTO1WRK+4+19(2),=C'#3' Set parm # indicator
         MVC   WTO1WRK+4+42(8),TCBADDR Copy TCB address
         LA    R4,WTO1WRK+4+25     Get target area address
VAL3LP   DS    ØH
         CLI   Ø(R3),X'ØØ'         End of data?
         BE    VAL3END             Yes - done with the data
         MVC   Ø(1,R4),Ø(R3)       Copy the next character
         LA    R3,1(,R3)           Point to next source byte
         LA    R4,1(,R4)           Point to next target byte
         B     VAL3LP              Check for more data
VAL3END  DS    ØH
         WTO   MF=(E,WTO1WRK)      Issue WTO
*********************************************************************
RETURN   DS    ØH
         L     R5,RETCODE          Load return code
         LR    R1,R13              Get temp storage address
         STORAGE RELEASE,LENGTH=WORKSIZE,ADDR=(R1)
         LR    R15,R5              Copy the return code
         PR Return
```

```
RETURNØØ DS    ØH
         MVC   RETCODE(4),=F'Ø'    Set return code value
         B     RETURN              Return
RETURNØ4 DS    ØH
         MVC   RETCODE(4),=F'4'    Set return code value
         B     RETURN              Return
*********************************************************************
WTO1LST  WTO   'ATTPGM1  - Parm value is xxxxxxxx for TCB xxxxxxxx. ', X
               ROUTCDE=(1),DESC=(6),MF=L
WTO1LN   EQU   *-WTO1LST
*********************************************************************
         LTORG ,
*
WORKAREA DSECT
SAVEAREA DS    18F
*
RETCODE  DS    F                   Return code
PARMADDR DS    F                   Parameter address
WTO1WRK  DS    ØD,CL(WTO1LN)       WTO work area
DBL1     DS    2D                  A work area
DBL2     DS    2D                  A work area
TCBADDR  DS    CL8                 TCB address
WORKSIZE EQU   *-WORKAREA
*********************************************************************
*
RØ       EQU   Ø
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R1Ø      EQU   1Ø
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         END
```

Rudy Douglas
System Programmer (Canada)

Splitting PDSs

Did you ever want to split off a portion of a PDS for testing? Or have you ever wanted to break up a PDS into a number of smaller PDSs so you could run multiple quick jobs instead of one long job?

The two biggest reasons I can remember for writing PDSSPLIT were using the CICS Load Module scanner (DFHEISUP) and AMBLIST against large PDSs. PDSSPLIT allowed me to get my results in a fraction of the time and avoided memory usage problems encountered with both programs. Since creating PDSSPLIT, I have found many other uses for it.

PDSSPLIT will generate new unique datasets for all the subsets and leave the original dataset intact. PDSSPLIT simply builds IEBCOPY control cards and invokes IEBCOPY to perform the copies. PDSSPLIT has three modes:

- ALPHA – create new datasets based on the first character of the member name.
- EVEN – create new datasets and evenly distribute the members across the number specified (default is 10).
- nn – create as many datasets as necessary to place *nn* members in each.

Sample JCL to run PDSSPLIT:

```
//jobcard…
//*****************************************************************
//* SPLIT A PDS                                                   *
//*                                                               *
//* OPTION=ALPHA     1 DSN FOR EACH UNIQUE FIRST CHARACTER OF     *
//*                  THE MEMBER NAME                              *
//* OPTION=EVEN      SPLIT INTO 1Ø EQUAL DSNS (SAME AS EVEN 1Ø)   *
//* OPTION='EVEN NN' SPLIT INTO 'NN' EQUAL DSNS                   *
//* OPTION=NN        SPLIT INTO DSNS EACH HOLDING NN MEMBERS      *
//*****************************************************************
//PDSSPLIT PROC
//PDSSPLIT EXEC PGM=IKJEFT01,DYNAMNBR=99,PARM='PDSSPLIT &OPTION'
//SYSEXEC  DD   DSN=yourid.EXEC,DISP=SHR
```

```
//SYSTSPRT DD   SYSOUT=*
//SYSTSIN  DD   DUMMY
//DIAGMSGS DD   SYSOUT=*
//INPUT    DD   DSN=&PDS,DISP=SHR
//SYSPRINT DD   SYSOUT=*
//         PEND
//ALPHA    EXEC PDSSPLIT,PDS=yourid.JCL,OPTION=ALPHA
//EVEN10   EXEC PDSSPLIT,PDS=yourid.JCL,OPTION=EVEN
//EVEN3    EXEC PDSSPLIT,PDS=yourid.JCL,OPTION='EVEN 3'
//P100     EXEC PDSSPLIT,PDS=yourid.JCL,OPTION=100
```

Here is sample program output for the ALPHA parameter:

```
JOBNAME ------- PDSSPLIT started 23 Sep 2004 22:54:30 on SY01 -------
JOBNAME

    6 $* members copied to YOURID.@001$006.YOURID.JCL RC=0
    1 #* members copied to YOURID.@002#001.YOURID.JCL RC=0
   25 @* members copied to YOURID.@003@025.YOURID.JCL RC=0
    7 A* members copied to YOURID.@004A007.YOURID.JCL RC=0
   20 B* members copied to YOURID.@005B020.YOURID.JCL RC=0
   44 C* members copied to YOURID.@006C044.YOURID.JCL RC=0
   48 D* members copied to YOURID.@007D048.YOURID.JCL RC=0
   13 E* members copied to YOURID.@008E013.YOURID.JCL RC=0
    8 F* members copied to YOURID.@009F008.YOURID.JCL RC=0
    6 G* members copied to YOURID.@010G006.YOURID.JCL RC=0
    4 H* members copied to YOURID.@011H004.YOURID.JCL RC=0
   24 I* members copied to YOURID.@012I024.YOURID.JCL RC=0
    6 J* members copied to YOURID.@013J006.YOURID.JCL RC=0
    1 K* members copied to YOURID.@014K001.YOURID.JCL RC=0
   16 L* members copied to YOURID.@015L016.YOURID.JCL RC=0
   20 M* members copied to YOURID.@016M020.YOURID.JCL RC=0
   19 N* members copied to YOURID.@017N019.YOURID.JCL RC=0
   15 P* members copied to YOURID.@018P015.YOURID.JCL RC=0
    3 Q* members copied to YOURID.@019Q003.YOURID.JCL RC=0
   24 R* members copied to YOURID.@020R024.YOURID.JCL RC=0
   50 S* members copied to YOURID.@021S050.YOURID.JCL RC=0
   27 T* members copied to YOURID.@022T027.YOURID.JCL RC=0
    2 U* members copied to YOURID.@023U002.YOURID.JCL RC=0
    4 V* members copied to YOURID.@024V004.YOURID.JCL RC=0
    7 W* members copied to YOURID.@025W007.YOURID.JCL RC=0
    4 X* members copied to YOURID.@026X004.YOURID.JCL RC=0
    1 Y* members copied to YOURID.@027Y001.YOURID.JCL RC=0
   24 Z* members copied to YOURID.@028Z024.YOURID.JCL RC=0

  429 members copied
   28 datasets created

 JOBNAME --- PDSSPLIT ended 23 Sep 2004 22:54:38 8.5 on SY01 RC=0 ----
JOBNAME
```

Here is sample output for the EVEN parameter:

```
JOBNAME ------- PDSSPLIT started 23 Sep 2004 22:54:38 on SY01 -------
JOBNAME

   43 members copied to YOURID.@001#043.YOURID.JCL RC=0
   43 members copied to YOURID.@002#043.YOURID.JCL RC=0
   43 members copied to YOURID.@003#043.YOURID.JCL RC=0
   43 members copied to YOURID.@004#043.YOURID.JCL RC=0
   43 members copied to YOURID.@005#043.YOURID.JCL RC=0
   43 members copied to YOURID.@006#043.YOURID.JCL RC=0
   43 members copied to YOURID.@007#043.YOURID.JCL RC=0
   43 members copied to YOURID.@008#043.YOURID.JCL RC=0
   43 members copied to YOURID.@009#043.YOURID.JCL RC=0
   42 members copied to YOURID.@010#042.YOURID.JCL RC=0

  429 members copied
   10 datasets created

 JOBNAME --- PDSSPLIT ended 23 Sep 2004 22:54:42 3.8 on SY01 RC=0 ----
JOBNAME
```

Here is sample output for the *nn* parameter:

```
JOBNAME ------- PDSSPLIT started 23 Sep 2004 22:54:44 on SY01 -------
JOBNAME

 100 members copied to YOURID.@001#100.YOURID.JCL RC=0
 100 members copied to YOURID.@002#100.YOURID.JCL RC=0
 100 members copied to YOURID.@003#100.YOURID.JCL RC=0
 100 members copied to YOURID.@004#100.YOURID.JCL RC=0
  29 members copied to YOURID.@005#029.YOURID.JCL RC=0

 429 members copied
   5 datasets created

JOBNAME --- PDSSPLIT ended 23 Sep 2004 22:54:47 2.5 on SY01 RC=0 ----
JOBNAME
```

CODE

```
/*********************************************************************/
/*                               REXX                                */
/*********************************************************************/
/* Purpose: Copy a PDS to multiple output PDSs based on criteria     */
/*-------------------------------------------------------------------*/
/* Syntax:  pdssplit criteria                                        */
/*-------------------------------------------------------------------*/
/* Parms: approach   - number or ALPHA or EVEN                       */
```

```
/*                                                                   */
/* Use a number to create multiple PDSs with that number of members  */
/* Use ALPHA to create a PDS for each letter of the alphabet         */
/* Use EVEN nn to create nn PDSs with an even portion of members     */
/*                                                                   */
/* Known issue: a single member copy does not work (creates PS DSN?) */
/*********************************************************************/
/*                       Change Log                                  */
/*                                                                   */
/*  Author       Date      Reason                                    */
/*  --------    ---------  ----------------------------------------- */
/*                                                                   */
/*********** @REFRESH BEGIN START    2004/03/06 13:16:32 *************/
/* Standard housekeeping activities                                  */
/*********************************************************************/
 call time 'r'
 parse arg parms
 signal on syntax name trap
 signal on failure name trap
 signal on novalue name trap
 probe = 'NONE'
 modtrace = 'NO'
 modspace = ''
 call stdentry 'DIAGMSGS'
 module = 'MAINLINE'
 push trace() time('L') module 'From:' 0 'Parms:' parms
 if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
 call modtrace 'START' 0
/*********************************************************************/
/* Set local estoeric names                                          */
/*********************************************************************/
 @vio   = 'VIO'
 @sysda = 'SYSDA'
/*********** @REFRESH END   START    2004/03/06 13:16:32 *************/
/* Accept and validate parms                                         */
/*********************************************************************/
 arg parm evencount .
 if parm = '' then
    parm = 'ALPHA'
 select
   when datatype(parm,'W') = 1 then parm = parm
   when parm = 'ALPHA' then parm = 'ALPHA'
   when parm = 'EVEN' then
        do
         parm = 'EVEN'
         if evencount = '' then evencount = 10
        end
   otherwise
      call rcexit 20 'Invalid parm:' parm', whole number, "ALPHA"',
          'or "EVEN"'
```

```
 end
 call saydd msgdd 1 'Parm used:' parm
/*********************************************************************/
/* Make sure the INPUT DD exists                                     */
/*********************************************************************/
 call ddcheck 'INPUT'
 source = sysdsname
/*********************************************************************/
/* Set initial defaults                                              */
/*********************************************************************/
 totcount = 0
 check = 1
 last = ''
 sets = -1
/*********************************************************************/
/* Get member names from the PDS                                     */
/*********************************************************************/
 call outtrap 'mem.'
"LISTDS '"source"' MEMBERS"
 x = outtrap('off')
/*********************************************************************/
/* Set option specific defaults                                      */
/*********************************************************************/
 select
   when parm = 'ALPHA' then mcount = 0
   when parm = 'EVEN' then
        do
         parm = format(((mem.0-7)/evencount),,0)
         mcount = parm + 1
        end
   otherwise mcount = parm + 1
 end
/*********************************************************************/
/* Process the members                                               */
/*********************************************************************/
 do i=7 to mem.0
    totcount = totcount + 1
    member = strip(mem.i)
/*********************************************************************/
/* Process ALPHA                                                     */
/*********************************************************************/
    if parm = 'ALPHA' then
       do
        char1 = substr(member,1,1)
        if char1 = last then
           do
            mcount = mcount + 1
            sysin.mcount = cont('                 'member',')
           end
        else
```

```
            do
             sets = sets + 1
             if i <> 7 then
                do
                 target = dsalloc(source right(sets,3,Ø),
                          right(mcount-1,3,Ø) parm mem.Ø-7)
                 say right(mcount-1,4) last'* members copied to' target,
                    'RC='copymem(source target)
                end
             sysin.1  = '  COPY INDD=SOURCE,OUTDD=TARGET'
             sysin.2  = cont('  SELECT MEMBER=('member',')
             last = char1
             mcount = 2
            end
       end
/*********************************************************************/
/* Process Numeric and EVEN                                          */
/*********************************************************************/
    if datatype(parm,'W') = 1 then
       do
        if mcount <= parm then
           do
            mcount = mcount + 1
            sysin.mcount = cont('                 'member',')
           end
        else
           do
            sets = sets + 1
            if i <> 7 then
               do
                target = dsalloc(source right(sets,3,Ø),
                          right(mcount-1,3,Ø) parm mem.Ø-7)
                say right(mcount-1,4) 'members copied to' target,
                   'RC='copymem(source target)
               end
            sysin.1  = '  COPY INDD=SOURCE,OUTDD=TARGET'
            sysin.2  = cont('  SELECT MEMBER=('member',')
            mcount = 2
           end
       end
 end
/*********************************************************************/
/* Process the last group                                            */
/*********************************************************************/
 sets = sets + 1
 target = dsalloc(source right(sets,3,Ø),
          right(mcount-1,3,Ø) parm mem.Ø-7)
 if parm = 'ALPHA' then
    say right(mcount-1,4) last'* members copied to' target,
       'RC='copymem(source target)
```

```
 else
    say right(mcount-1,4) 'members copied to' target,
       'RC='copymem(source target)
/*********************************************************************/
/* Print stats                                                       */
/*********************************************************************/
 say
 say right(totcount,4) 'members copied'
 say right(sets,4) 'datasets created'
/*********************************************************************/
/* Shutdown                                                          */
/*********************************************************************/
 shutdown: nop
/*********************************************************************/
/* Put unique shutdown logic before the call to stdexit              */
/*********** @REFRESH BEGIN STOP     2002/08/03 08:42:33 *************/
/* Shutdown message and terminate                                    */
/*********************************************************************/
           call stdexit time('e')
/*********** @REFRESH END   STOP     2002/08/03 08:42:33 *************/
/* Internal Subroutines - not refreshable                            */
/*                                                                   */
/* DSALLOC  - Allocate the new DSN                                   */
/* COPYMEM  - Invoke IEBCOPY to copy members                         */
/* CONT     - Append line with a continuation in column 72           */
/*                                                                   */
/*********************************************************************/
/* DSALLOC  - Allocate the new DSN                                   */
/*********************************************************************/
 dsalloc: arg olddsn num memnum parm totmem
          if parm = 'ALPHA' then
             clonetype = last
          else
             clonetype = '#'
          newdsn = userid()'.@'num||clonetype||memnum'.'olddsn
          dirblks = format(memnum/4,,0)
          pmem = memnum/totmem
          prispace = format(pmem*(sysused/sysblkstrk/systrkscyl),,0)
          if prispace = 0 then prispace = 1
          secspace = prispace * 5
          if sysrecfm = 'U' then
             do
              call tsotrap "ALLOC F(CLONE) DA("qdsn(newdsn)")",
                           "LIKE("qdsn(olddsn)") DIR("dirblks")",
                           "SPACE("prispace secspace") CYLINDERS",
                           "BLKSIZE("sysblksize")"
             end
           else
             do
              call tsotrap "ALLOC F(CLONE) DA("qdsn(newdsn)")",
```

```
                          "LIKE("qdsn(olddsn)") DIR("dirblks")",
                          "SPACE("prispace secspace") CYLINDERS",
                          "LRECL("syslrecl") BLKSIZE("sysblksize")"
            end
          call tsotrap "FREE F(CLONE)"
          return newdsn
/*********************************************************************/
/* COPYMEM  - Invoke IEBCOPY to copy members                         */
/*********************************************************************/
 copymem: arg olddsn newdsn
          close = mcount + 1
          sysin.close = '                )'
          call tsotrap "ALLOC F(SOURCE) DA("qdsn(olddsn)") SHR REUSE"
          call tsotrap "ALLOC F(TARGET) DA("qdsn(newdsn)") SHR REUSE"
          call viodd 'SYSIN'
          address TSO "CALL *(IEBCOPY)"
          call rcexit RC 'IEBCOPY error copying' olddsn 'to' newdsn
          call tsotrap "FREE F(SOURCE TARGET SYSIN)"
          drop sysin.
          return 0
/*********************************************************************/
/* CONT     - Append line with a continuation in column 72           */
/*********************************************************************/
 cont: parse arg string
       contstring = string copies(' ',70-length(string))||'X'
       return contstring
/*********** @REFRESH BEGIN SUBBOX   2004/03/10 01:25:03 *************/
/*                                                                   */
/* 20 Internal Subroutines provided in PDSSPLIT                      */
/*                                                                   */
/* Last Subroutine REFRESH was 29 Jul 2004 20:55:33                  */
/*                                                                   */
/* RCEXIT   - Exit on non-zero return codes                          */
/* TRAP     - Issue a common trap error message using rcexit         */
/* ERRMSG   - Build common error message with failing line number    */
/* STDENTRY - Standard Entry logic                                   */
/* STDEXIT  - Standard Exit logic                                    */
/* MSG      - Determine whether to SAY or ISPEXEC SETMSG the message */
/* DDCHECK  - Determine whether a required DD is allocated           */
/* DDLIST   - Returns number of DDs and populates DDLIST variable    */
/* DDDSNS   - Returns number of DSNs in a DD and populates DDDSNS    */
/* QDSN     - Make sure there are only one set of quotes             */
/* TSOTRAP  - Capture the output from a TSO command in a stem        */
/* SAYDD    - Print messages to the requested DD                     */
/* JOBINFO  - Get job-related data from control blocks               */
/* PTR      - Pointer to a storage location                          */
/* STG      - Return the data from a storage location                */
/* VIODD    - EXECIO a stem into a sequential dataset                */
/* MODTRACE - Module Trace                                           */
/*                                                                   */
```

```
/*********** @REFRESH END   SUBBOX   2004/03/10 01:25:03 *************/
/*********** @REFRESH BEGIN RCEXIT   2003/05/14 12:24:50 *************/
/* RCEXIT   - Exit on non-zero return codes                          */
/*-------------------------------------------------------------------*/
/* EXITRC   - Return code to exit with (if non zero)                 */
/* ZEDLMSG  - Message text for it with for non zero EXITRC's         */
/*********************************************************************/
 rcexit: parse arg EXITRC zedlmsg
         if EXITRC <> 0 then
            do
             trace 'o'
/*********************************************************************/
/* If execution environment is ISPF then VPUT ZISPFRC                */
/*********************************************************************/
             if execenv = 'TSO' | execenv = 'ISPF' then
                do
                 if ispfenv = 'YES' then
                    do
                     zispfrc = EXITRC
/*********************************************************************/
/* Does not call ISPWRAP to avoid obscuring error message modules    */
/*********************************************************************/
                      address ISPEXEC "VPUT (ZISPFRC)"
                    end
                end
/*********************************************************************/
/* If a message is provided, wrap it in date, time and EXITRC        */
/*********************************************************************/
             if zedlmsg <> '' then
                do
                 zedlmsg = time('L') execname zedlmsg 'RC='EXITRC
                 call msg zedlmsg
                end
/*********************************************************************/
/* Write the contents of the Parentage Stack                         */
/*********************************************************************/
             stacktitle = 'Parentage Stack Trace ('queued()' entries):'
/*********************************************************************/
/* Write to MSGDD if background and MSGDD exists                     */
/*********************************************************************/
             if tsoenv = 'BACK' then
                do
                 if subword(zedlmsg,9,1) = msgdd then
                    do
                     say zedlmsg
                     signal shutdown
                    end
                 else
                    do
                     call saydd msgdd 1 zedlmsg
```

```
                    call saydd msgdd 1 stacktitle
                   end
               end
            else
/*********************************************************************/
/* Write to the ISPF Log if foreground                               */
/*********************************************************************/
               do
                zerrlm = zedlmsg
                address ISPEXEC "LOG MSG(ISRZØØ3)"
                zerrlm = center(' 'stacktitle' ',78,'-')
                address ISPEXEC "LOG MSG(ISRZØØ3)"
               end
/*********************************************************************/
/* Unload the Parentage Stack                                        */
/*********************************************************************/
            do queued()
               pull stackinfo
               if tsoenv = 'BACK' then
                  do
                   call saydd msgdd Ø stackinfo
                  end
               else
                  do
                   zerrlm = stackinfo
                   address ISPEXEC "LOG MSG(ISRZØØ3)"
                  end
            end
/*********************************************************************/
/* Put a terminator in the ISPF Log for the Parentage Stack          */
/*********************************************************************/
            if tsoenv = 'FORE' then
               do
                zerrlm = center(' 'stacktitle' ',78,'-')
                address ISPEXEC "LOG MSG(ISRZØØ3)"
               end
/*********************************************************************/
/* Signal SHUTDOWN.  SHUTDOWN label MUST exist in the program        */
/*********************************************************************/
            signal shutdown
           end
         else
            return
/*********** @REFRESH END   RCEXIT   2ØØ3/Ø5/14 12:24:5Ø *************/
/*********** @REFRESH BEGIN TRAP     2ØØ2/Ø8/Ø7 11:48:14 *************/
/* TRAP     - Issue a common trap error message using rcexit         */
/*-------------------------------------------------------------------*/
/* PARM     - N/A                                                    */
/*********************************************************************/
 trap: traptype = condition('C')
```

```
       if traptype = 'SYNTAX' then
          msg = errortext(RC)
       else
          msg = condition('D')
       trapline = strip(sourceline(sigl))
       msg = traptype 'TRAP:' msg', Line:' sigl '"'trapline'"'
       call rcexit 666 msg
/*********** @REFRESH END   TRAP     2002/08/07 11:48:14 *************/
/*********** @REFRESH BEGIN ERRMSG   2002/08/10 16:53:04 *************/
/* ERRMSG   - Build common error message with failing line number    */
/*-------------------------------------------------------------------*/
/* ERRLINE  - The failing line number passed by caller from SIGL     */
/* TEXT     - Error message text passed by caller                    */
/*********************************************************************/
 errmsg: nop
         parse arg errline text
         return 'Error on statement' errline',' text
/*********** @REFRESH END   ERRMSG   2002/08/10 16:53:04 *************/
/*********** @REFRESH BEGIN STDENTRY 2004/04/07 19:17:48 *************/
/* STDENTRY - Standard Entry logic                                   */
/*-------------------------------------------------------------------*/
/* MSGDD    - Optional MSGDD used only in background                 */
/*********************************************************************/
 stdentry: module = 'STDENTRY'
           if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
           parse arg sparms
           push trace() time('L') module 'From:' sigl 'Parms:' sparms
           arg msgdd
           parse upper source . . execname . execdsn . . execenv .
/*********************************************************************/
/* Start-up values                                                   */
/*********************************************************************/
           EXITRC = 0
           MAXRC = 0
           ispfenv = 'NO'
           popup = 'NO'
           lockpop = 'NO'
           headoff = 'NO'
           hcreator = 'NO'
           keepstack = 'NO'
           lpar = mvsvar('SYSNAME')
           zedlmsg = 'Default shutdown message'
/*********************************************************************/
/* Determine environment                                             */
/*********************************************************************/
           if substr(execenv,1,3) <> 'TSO' & execenv <> 'ISPF' then
              tsoenv = 'NONE'
           else
              do
               tsoenv = sysvar('SYSENV')
```

```
               signal off failure
              "ISPQRY"
               ISPRC = RC
               if ISPRC = Ø then
                  do
                   ispfenv = 'YES'
/*********************************************************************/
/* Check if HEADING ISPF table exists already, if so set HEADOFF=YES */
/*********************************************************************/
                   call ispwrap "VGET (ZSCREEN)"
                   if tsoenv = 'BACK' then
                      htable = jobinfo(1)||jobinfo(2)
                   else
                      htable = userid()||zscreen
                   TBCRC = ispwrap(8 "TBCREATE" htable "KEYS(HEAD)")
                   if TBCRC = Ø then
                      do
                       headoff = 'NO'
                       hcreator = 'YES'
                      end
                   else
                      do
                       headoff = 'YES'
                      end
                  end
               signal on failure name trap
              end
/*********************************************************************/
/* MODTRACE must occur after the setting of ISPFENV                  */
/*********************************************************************/
           call modtrace 'START' sigl
/*********************************************************************/
/* Start-up message (if batch)                                       */
/*********************************************************************/
           startmsg = execname 'started' date() time() 'on' lpar
           if tsoenv = 'BACK' & sysvar('SYSNEST') = 'NO' &,
              headoff = 'NO' then
              do
               jobname = mvsvar('SYMDEF','JOBNAME')
               jobinfo = jobinfo()
               parse var jobinfo jobtype jobnum .
               say jobname center(' 'startmsg' ',61,'-') jobtype jobnum
               say
               if ISPRC = -3 then
                  do
                   call saydd msgdd 1 'ISPF ISPQRY module not found,',
                                      'ISPQRY is usually in the LINKLST'
                   call rcexit 2Ø 'ISPF ISPQRY module is missing'
                  end
/*********************************************************************/
```

```
/* If MSGDD is provided, write the STARTMSG and SYSEXEC DSN to MSGDD */
/*********************************************************************/
               if msgdd <> '' then
                  do
                   call ddcheck msgdd
                   call saydd msgdd 1 startmsg
                   call ddcheck 'SYSEXEC'
                   call saydd msgdd Ø execname 'loaded from' sysdsname
/*********************************************************************/
/* If there are PARMS, write them to the MSGDD                       */
/*********************************************************************/
                   if parms <> '' then
                      call saydd msgdd Ø 'Parms:' parms
/*********************************************************************/
/* If there is a STEPLIB, write the STEPLIB DSN MSGDD                */
/*********************************************************************/
                   if listdsi('STEPLIB' 'FILE') = Ø then
                      do
                       steplibs = dddsns('STEPLIB')
                       call saydd msgdd Ø 'STEPLIB executables loaded',
                           'from' word(dddsns,1)
                       if dddsns('STEPLIB') > 1 then
                          do
                           do stl=2 to steplibs
                              call saydd msgdd Ø copies(' ',31),
                                   word(dddsns,stl)
                           end
                          end
                      end
                  end
               end
/*********************************************************************/
/* If foreground, save ZFKA and turn off the FKA display             */
/*********************************************************************/
            else
               do
                fkaset = 'OFF'
                call ispwrap "VGET (ZFKA) PROFILE"
                if zfka <> 'OFF' & tsoenv = 'FORE' then
                   do
                    fkaset = zfka
                    fkacmd = 'FKA OFF'
                    call ispwrap "CONTROL DISPLAY SAVE"
                   call ispwrap "DISPLAY PANEL(ISPBLANK) COMMAND(FKACMD)"
                    call ispwrap "CONTROL DISPLAY RESTORE"
                   end
               end
/*********************************************************************/
            pull tracelvl . module . sigl . sparms
            call modtrace 'STOP' sigl
```

```
           interpret 'trace' tracelvl
           return
/*********** @REFRESH END   STDENTRY 2004/04/07 19:17:48 *************/
/*********** @REFRESH BEGIN STDEXIT  2003/11/16 22:46:29 *************/
/* STDEXIT  - Standard Exit logic                                    */
/*-------------------------------------------------------------------*/
/* ENDTIME  - Elapsed time                                           */
/* Note: Caller must set KEEPSTACK if the stack is valid             */
/*********************************************************************/
 stdexit: module = 'STDEXIT'
          if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg endtime
          endmsg = execname 'ended' date() time() format(endtime,,1)
/*********************************************************************/
/* if MAXRC is greater then EXITRC then set EXITRC to MAXRC          */
/*********************************************************************/
          if MAXRC > EXITRC then EXITRC = MAXRC
          endmsg = endmsg 'on' lpar 'RC='EXITRC
          if tsoenv = 'BACK' & sysvar('SYSNEST') = 'NO' &,
             headoff = 'NO' then
             do
              say
              say jobname center(' 'endmsg' ',61,'-') jobtype jobnum
/*********************************************************************/
/* Make sure this isn't a MSGDD missing error then log to MSGDD      */
/*********************************************************************/
              if msgdd <> '' & subword(zedlmsg,9,1) <> msgdd then
                 do
                  call saydd msgdd 1 execname 'ran in' endtime 'seconds'
                  call saydd msgdd 0 endmsg
                 end
             end
/*********************************************************************/
/* If foreground, reset the FKA if necessary                         */
/*********************************************************************/
          else
             do
              if fkaset <> 'OFF' then
                 do
                  fkafix = 'FKA'
                  call ispwrap "CONTROL DISPLAY SAVE"
                  call ispwrap "DISPLAY PANEL(ISPBLANK) COMMAND(FKAFIX)"
                  if fkaset = 'SHORT' then
                     call ispwrap "DISPLAY PANEL(ISPBLANK)",
                                  "COMMAND(FKAFIX)"
                  call ispwrap "CONTROL DISPLAY RESTORE"
                 end
```

```
            end
/*********************************************************************/
/* Clean up the temporary HEADING table                              */
/*********************************************************************/
          if ispfenv = 'YES' & hcreator = 'YES' then
             call ispwrap "TBEND" htable
/*********************************************************************/
/* Remove STDEXIT and MAINLINE Parentage Stack entries, if there     */
/*********************************************************************/
          call modtrace 'STOP' sigl
          if queued() > Ø then pull . . module . sigl . sparms
          if queued() > Ø then pull . . module . sigl . sparms
          if tsoenv = 'FORE' & queued() > Ø & keepstack = 'NO' then
             pull . . module . sigl . sparms
/*********************************************************************/
/* if the Parentage Stack is not empty, display its contents         */
/*********************************************************************/
          if queued() > Ø & keepstack = 'NO' then
             do
              say queued() 'Leftover Parentage Stack Entries:'
              say
              do queued()
                 pull stackundo
                 say stackundo
              end
              EXITRC = 1
             end
/*********************************************************************/
/* Exit                                                              */
/*********************************************************************/
          exit(EXITRC)
/*********** @REFRESH END   STDEXIT  2003/11/16 22:46:29 *************/
/*********** @REFRESH BEGIN MSG      2002/09/11 01:35:53 *************/
/* MSG      - Determine whether to SAY or ISPEXEC SETMSG the message */
/*-------------------------------------------------------------------*/
/* ZEDLMSG  - The long message variable                              */
/*********************************************************************/
 msg: module = 'MSG'
      parse arg zedlmsg
      if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
      parse arg sparms
      push trace() time('L') module 'From:' sigl 'Parms:' sparms
      call modtrace 'START' sigl
/*********************************************************************/
/* If this is background or OMVS use SAY                             */
/*********************************************************************/
      if tsoenv = 'BACK' | execenv = 'OMVS' then
         say zedlmsg
      else
/*********************************************************************/
```

```
/* If this is foreground and ISPF is available, use SETMSG           */
/*********************************************************************/
         do
          if ispfenv = 'YES' then
/*********************************************************************/
/* Does not call ISPWRAP to avoid obscuring error message modules    */
/*********************************************************************/
             address ISPEXEC "SETMSG MSG(ISRZ000)"
          else
             say zedlmsg
         end
      pull tracelvl . module . sigl . sparms
      call modtrace 'STOP' sigl
      interpret 'trace' tracelvl
      return
/*********** @REFRESH END   MSG      2002/09/11 01:35:53 *************/
/*********** @REFRESH BEGIN DDCHECK  2002/09/11 01:08:30 *************/
/* DDCHECK  - Determine if a required DD is allocated                */
/*-------------------------------------------------------------------*/
/* DD       - DDNAME to confirm                                      */
/*********************************************************************/
 ddcheck: module = 'DDCHECK'
          if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg dd
          dderrmsg = 'OK'
          LRC = listdsi(dd "FILE")
/*********************************************************************/
/* Allow sysreason=3 to verify SYSOUT DD statements                  */
/*********************************************************************/
          if LRC <> 0 & strip(sysreason,'L',0) <> 3 then
             do
              dderrmsg = errmsg(sigl 'Required DD' dd 'is missing')
              call rcexit LRC dderrmsg sysmsglvl2
             end
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return
/*********** @REFRESH END   DDCHECK  2002/09/11 01:08:30 *************/
/*********** @REFRESH BEGIN DDLIST   2002/12/15 04:54:32 *************/
/* DDLIST   - Returns number of DDs and populates DDLIST variable    */
/*-------------------------------------------------------------------*/
/* N/A      - None                                                   */
/*********************************************************************/
 ddlist: module = 'DDLIST'
         if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
         parse arg sparms
```

```
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
/*********************************************************************/
/* Trap the output from the LISTA STATUS command                     */
/*********************************************************************/
         call outtrap 'lines.'
         address TSO "LISTALC STATUS"
         call outtrap 'off'
         ddnum = 0
/*********************************************************************/
/* Parse out the DDNAMEs and concatenate into a list                 */
/*********************************************************************/
         ddlist = ''
         do ddl=1 to lines.0
            if words(lines.ddl) = 2 then
               do
                parse upper var lines.ddl ddname .
                ddlist = ddlist ddname
                ddnum = ddnum + 1
               end
            else
               do
                iterate
               end
         end
/*********************************************************************/
/* Return the number of DDs                                          */
/*********************************************************************/
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return ddnum
/*********** @REFRESH END   DDLIST   2002/12/15 04:54:32 *************/
/*********** @REFRESH BEGIN DDDSNS   2002/09/11 00:37:36 *************/
/* DDDSNS   - Returns number of DSNs in a DD and populates DDDSNS    */
/*-------------------------------------------------------------------*/
/* TARGDD   - DD to return DSNs for                                  */
/*********************************************************************/
 dddsns: module = 'DDDSNS'
         if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
         arg targdd
         if targdd = '' then call rcexit 77 'DD missing for DDDSNS'
/*********************************************************************/
/* Trap the output from the LISTA STATUS command                     */
/*********************************************************************/
         x = outtrap('lines.')
         address TSO "LISTALC STATUS"
```

```
         dsnnum = Ø
         ddname = '$DDNAME$'
/*********************************************************************/
/* Parse out the DDNAMEs, locate the target DD and concatentate DSNs */
/*********************************************************************/
         do ddd=1 to lines.Ø
            select
               when words(lines.ddd) = 1 & targdd = ddname &,
                    lines.ddd <> 'KEEP' then
                    dddsns = dddsns strip(lines.ddd)
               when words(lines.ddd) = 1 & strip(lines.ddd),
                    <> 'KEEP' then
                    dddsn.ddd = strip(lines.ddd)
               when words(lines.ddd) = 2 then
                    do
                     parse upper var lines.ddd ddname .
                     if targdd = ddname then
                        do
                         fdsn = ddd - 1
                         dddsns = lines.fdsn
                        end
                    end
               otherwise iterate
            end
         end
/*********************************************************************/
/* Get the last DD                                                   */
/*********************************************************************/
         ddnum = ddlist()
         lastdd = word(ddlist,ddnum)
/*********************************************************************/
/* Remove the last DSN from the list if not the last DD or SYSEXEC   */
/*********************************************************************/
         if targdd <> 'SYSEXEC' & targdd <> lastdd then
            do
             dsnnum = words(dddsns) - 1
             dddsns = subword(dddsns,1,dsnnum)
            end
/*********************************************************************/
/* Return the number of DSNs in the DD                               */
/*********************************************************************/
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return dsnnum
/*********** @REFRESH END   DDDSNS   2002/09/11 00:37:36 *************/
/*********** @REFRESH BEGIN QDSN     2002/09/11 01:15:23 *************/
/* QDSN     - Make sure there are only one set of quotes             */
/*-------------------------------------------------------------------*/
/* QDSN     - The DSN                                                */
```

```
/*********************************************************************/
 qdsn: module = 'QDSN'
       if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
       parse arg sparms
       push trace() time('L') module 'From:' sigl 'Parms:' sparms
       call modtrace 'START' sigl
       parse arg qdsn
       qdsn = "'"strip(qdsn,"B","'")"'"
       pull tracelvl . module . sigl . sparms
       call modtrace 'STOP' sigl
       interpret 'trace' tracelvl
       return qdsn
/*********** @REFRESH END   QDSN     2ØØ2/Ø9/11 Ø1:15:23 *************/
/*********** @REFRESH BEGIN TSOTRAP  2ØØ2/12/15 Ø5:18:45 *************/
/* TSOTRAP  - Capture the output from a TSO command in a stem        */
/*-------------------------------------------------------------------*/
/* VALIDRC  - Optional valid RC, defaults to zero                    */
/* TSOPARM  - Valid TSO command                                      */
/*********************************************************************/
 tsotrap: module = 'TSOTRAP'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          parse arg tsoparm
/*********************************************************************/
/* If the optional valid_rc parm is present use it, if not assume Ø  */
/*********************************************************************/
          parse var tsoparm valid_rc tso_cmd
          if datatype(valid_rc,'W') = Ø then
             do
              valid_rc = Ø
              tso_cmd = tsoparm
             end
          call outtrap 'tsoout.'
          tsoline = sigl
          address TSO tso_cmd
          CRC = RC
          call outtrap 'off'
/*********************************************************************/
/* If RC = Ø then return                                             */
/*********************************************************************/
          if CRC <= valid_rc then
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return CRC
             end
          else
```

```
            do
             trapmsg = center(' TSO Command Error Trap ',78,'-')
             terrmsg = errmsg(sigl 'TSO Command:')
/*********************************************************************/
/* If RC <> Ø then format output depending on environment            */
/*********************************************************************/
             if tsoenv = 'BACK' | execenv = 'OMVS' then
                do
                 say trapmsg
                 do c=1 to tsoout.Ø
                    say tsoout.c
                 end
                 say trapmsg
                 call rcexit CRC terrmsg tso_cmd
                end
             else
/*********************************************************************/
/* If this is foreground and ISPF is available, use the ISPF LOG     */
/*********************************************************************/
                do
                 if ispfenv = 'YES' then
                    do
                     zedlmsg = trapmsg
/*********************************************************************/
/* Does not call ISPWRAP to avoid obscuring error message modules    */
/*********************************************************************/
                     address ISPEXEC "LOG MSG(ISRZØØØ)"
                     do c=1 to tsoout.Ø
                        zedlmsg = tsoout.c
                        address ISPEXEC "LOG MSG(ISRZØØØ)"
                     end
                     zedlmsg = trapmsg
                     address ISPEXEC "LOG MSG(ISRZØØØ)"
                     call rcexit CRC terrmsg tso_cmd,
                          ' see the ISPF Log (Option 7.5) for details'
                    end
                 else
                    do
                     say trapmsg
                     do c=1 to tsoout.Ø
                        say tsoout.c
                     end
                     say trapmsg
                     call rcexit CRC terrmsg tso_cmd
                    end
                end
            end
/*********** @REFRESH END   TSOTRAP  2ØØ2/12/15 Ø5:18:45 *************/
/*********** @REFRESH BEGIN SAYDD    2ØØ4/Ø3/29 23:48:37 *************/
/* SAYDD    - Print messages to the requested DD                     */
```

```
/*-------------------------------------------------------------------*/
/* MSGDD    - DDNAME to write messages to                            */
/* MSGLINES - number of blank lines to put before and after          */
/* MESSAGE  - Text to write to the MSGDD                             */
/*-------------------------------------------------------------------*/
 saydd: module = 'SAYDD'
        if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
        parse arg sparms
        push trace() time('L') module 'From:' sigl 'Parms:' sparms
        call modtrace 'START' sigl
        parse arg msgdd msglines message
        if words(msgdd msglines message) < 3 then
           call rcexit 33 'Missing MSGDD or MSGLINES'
        if datatype(msglines) <> 'NUM' then
           call rcexit 34 'MSGLINES must be numeric'
/*-------------------------------------------------------------------*/
/* If this is not background then bypass                             */
/*-------------------------------------------------------------------*/
        if tsoenv <> 'BACK' then
           do
            pull tracelvl . module . sigl . sparms
            call modtrace 'STOP' sigl
            interpret 'trace' tracelvl
            return
           end
/*-------------------------------------------------------------------*/
/* Confirm the MSGDD exists                                          */
/*-------------------------------------------------------------------*/
        call ddcheck msgdd
/*-------------------------------------------------------------------*/
/* If a number is provided, add that number of blank lines before    */
/* the message                                                       */
/*-------------------------------------------------------------------*/
        msgb = 1
        if msglines > 0 then
           do msgb=1 to msglines
              msgline.msgb = ' '
           end
/*-------------------------------------------------------------------*/
/* If the linesize is too long break it into multiple lines and      */
/* create continuation records                                       */
/*-------------------------------------------------------------------*/
        msgm = msgb
        if length(message) > 60 & substr(message,1,2) <> '@@' then
           do
            messst = lastpos(' ',message,60)
            messseg = substr(message,1,messst)
            msgline.msgm = date() time() strip(messseg)
            message = strip(delstr(message,1,messst))
            do while length(message) > 0
```

```
               msgm = msgm + 1
               if length(message) > 55 then
                  messt = lastpos(' ',message,55)
               if messt > Ø then
                  messseg = substr(message,1,messt)
               else
                  messseg = substr(message,1,length(message))
               msgline.msgm = date() time() 'CONT:' strip(messseg)
               message = strip(delstr(message,1,length(messseg)))
            end
           end
        else
/*********************************************************************/
/* Build print lines. Default strips and prefixes date and timestamp */
/* @BLANK - Blank line, no date and timestamp                        */
/* @      - No stripping, retains leading blanks                     */
/* @@     - No stripping, No date and timestamp                      */
/*********************************************************************/
           do
            select
               when message = '@BLANK@' then msgline.msgm = ' '
               when word(message,1) = '@' then
                    do
                     message = substr(message,2,length(message)-1)
                     msgline.msgm = date() time() message
                    end
               when substr(message,1,2) = '@@' then
                    do
                     message = substr(message,3,length(message)-2)
                     msgline.msgm = message
                    end
               otherwise msgline.msgm = date() time() strip(message)
            end
           end
/*********************************************************************/
/* If a number is provided, add that number of blank lines after     */
/* the message                                                       */
/*********************************************************************/
        if msglines > Ø then
           do msgt=1 to msglines
              msge = msgt + msgm
              msgline.msge = ' '
           end
/*********************************************************************/
/* Write the contents of the MSGLINE stem to the MSGDD               */
/*********************************************************************/
        call tsotrap "EXECIO * DISKW" msgdd "(STEM MSGLINE. FINIS"
        drop msgline. msgb msgt msge
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
```

```
        interpret 'trace' tracelvl
        return
/*********** @REFRESH END   SAYDD    2004/03/29 23:48:37 *************/
/*********** @REFRESH BEGIN JOBINFO  2002/09/11 01:12:59 *************/
/* JOBINFO  - Get job related data from control blocks               */
/*------------------------------------------------------------------*/
/* ITEM     - Optional item number desired, default is all          */
/********************************************************************/
 jobinfo: module = 'JOBINFO'
          if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg item
/********************************************************************/
/* Chase control blocks                                             */
/********************************************************************/
          tcb      = ptr(540)
          ascb     = ptr(548)
          tiot     = ptr(tcb+12)
          jscb     = ptr(tcb+180)
          ssib     = ptr(jscb+316)
          asid     = c2d(stg(ascb+36,2))
          jobtype  = stg(ssib+12,3)
          jobnum   = strip(stg(ssib+15,5),'L',0)
          stepname = stg(tiot+8,8)
          procstep = stg(tiot+16,8)
          program  = stg(jscb+360,8)
          jobdata  = jobtype jobnum stepname procstep program asid
/********************************************************************/
/* Return job data                                                  */
/********************************************************************/
          if item <> '' & (datatype(item,'W') = 1) then
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return word(jobdata,item)
             end
          else
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return jobdata
             end
/*********** @REFRESH END   JOBINFO  2002/09/11 01:12:59 *************/
/*********** @REFRESH BEGIN PTR      2002/07/13 15:45:36 *************/
/* PTR      - Pointer to a storage location                         */
/*------------------------------------------------------------------*/
```

```
/* ARG(1)   - Storage Address                                          */
/*********************************************************************/
 ptr: return c2d(storage(d2x(arg(1)),4))
/*********** @REFRESH END   PTR      2002/07/13 15:45:36 *************/
/*********** @REFRESH BEGIN STG      2002/07/13 15:49:12 *************/
/* STG      - Return the data from a storage location                  */
/*-------------------------------------------------------------------*/
/* ARG(1)   - Location                                                 */
/* ARG(2)   - Length                                                   */
/*********************************************************************/
 stg: return storage(d2x(arg(1)),arg(2))
/*********** @REFRESH END   STG      2002/07/13 15:49:12 *************/
/*********** @REFRESH BEGIN VIODD    2004/06/08 11:17:36 *************/
/* VIODD    - EXECIO a stem into a sequential dataset                  */
/*-------------------------------------------------------------------*/
/* VIODD    - The member to create                                     */
/* VIOLRECL - The LRECL for the VIODD (defaults to 80)                 */
/*********************************************************************/
 viodd: module = 'VIODD'
        if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
        parse arg sparms
        push trace() time('L') module 'From:' sigl 'Parms:' sparms
        call modtrace 'START' sigl
        arg viodd violrecl viorecfm
        if viodd = '' then call rcexit 88 'VIODD missing'
        if violrecl = '' then violrecl = 80
        if viorecfm = '' then viorecfm = 'F B'
/*********************************************************************/
/* If DD exists, FREE it                                             */
/*********************************************************************/
        if listdsi(viodd 'FILE') = 0 then
           call tsotrap "FREE F("viodd")"
/*********************************************************************/
/* ALLOCATE a VIO DSN                                                */
/*********************************************************************/
        call tsotrap "ALLOC F("viodd") UNIT("@vio") SPACE(1 5)",
                     "LRECL("violrecl") BLKSIZE(0) REUSE",
                     "RECFM("viorecfm") CYLINDERS"
/*********************************************************************/
/* Write the stem variables into the VIO DSN                         */
/*********************************************************************/
        call tsotrap "EXECIO * DISKW" viodd "(STEM" viodd". FINIS"
/*********************************************************************/
/* DROP the stem variable                                            */
/*********************************************************************/
        interpret 'drop' viodd'.'
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return
```

```
/*********** @REFRESH END   VIODD    2004/06/08 11:17:36 *************/
/*********** @REFRESH BEGIN MODTRACE 2003/12/31 21:56:54 *************/
/* MODTRACE - Module Trace                                           */
/*-------------------------------------------------------------------*/
/* TRACETYP - Type of trace entry                                    */
/* SIGLINE  - The line number called from                            */
/*********************************************************************/
 modtrace: if modtrace = 'NO' then return
           arg tracetyp sigline
           tracetyp = left(tracetyp,5)
           sigline = left(sigline,5)
/*********************************************************************/
/* Adjust MODSPACE for START                                         */
/*********************************************************************/
           if tracetyp = 'START' then
              modspace = substr(modspace,1,length(modspace)+1)
/*********************************************************************/
/* Set the trace entry                                               */
/*********************************************************************/
           traceline = modspace time('L') tracetyp module sigline sparms
/*********************************************************************/
/* Adjust MODSPACE for STOP                                          */
/*********************************************************************/
           if tracetyp = 'STOP' then
              modspace = substr(modspace,1,length(modspace)-1)
/*********************************************************************/
/* Determine where to write the traceline                            */
/*********************************************************************/
           if ispfenv = 'YES' & tsoenv = 'FORE' then
/*********************************************************************/
/* Write to the ISPF Log, do not use ISPWRAP here                    */
/*********************************************************************/
              do
               zedlmsg = traceline
               address ISPEXEC "LOG MSG(ISRZ000)"
              end
           else
              say traceline
/*********************************************************************/
/* SAY to SYSTSPRT                                                   */
/*********************************************************************/
           return
/*********** @REFRESH END   MODTRACE 2003/12/31 21:56:54 *************/
```

Robert Zenuk
Systems Programmer (USA)

TPS Systems has announced TPS/JES Services, which is aimed at companies looking to consolidate older communications infrastructure to take advantage of newer multi-protocol communication technologies.

TPS/JES Services comprises two components – TPS/JES Services Server and TPS/RJS (Remote JES Services) Client. The Server operates as a z/OS component executing in the background to make JES2 and JES3 available to a TCP/IP-based client. TPS/RJS allows a multitude of simultaneous client connections, while maintaining only a single instance of the Server. The Server module interfaces with the JES system utilizing the z/OS SSI-based SAPI interface to become an external writer with the ability to act as a 'hot writer'; automatically processing JES SYSOUT output as it becomes available. The TCP/IP client/server connection protocol used between the TPS/JES Services Server and the TPS/RJS Client include features that prevent unauthorized access, as well as compression and optional SSL encryption capability.

For further information contact:
TPS Systems, 14100 San Pedro Avenue, Suite 600, San Antonio, TX 78232-4399, USA.
Tel: (210) 496 1984.
URL: www.tps.com/jes_o.html.

***

Phoenix Software has announced Version 4.1.0 of (E)JES, its systems management tool that provides users with information to monitor, manage, and control their z/OS JESplex

The new version provides a single look-and-feel to users of both JES2 and JES3.

With (E)JES, users can: control job processing (hold, release, cancel, and purge jobs); monitor jobs while they are running; browse jobs without printing, control JESplex parameters, JES-managed initiators (JES2), job classes, and job class groups (JES3); control printers, punches, functional subsystems (JES3), and NJE resources; control JES spool configuration.; control WLM scheduling environments and resources; control WLM enclaves and OMVS processes running under z/OS Unix System Services; and issue system commands that affect jobs.

For further information contact:
Phoenix Software International, 5200 West Century Boulevard, Suite 800, Los Angeles, CA 90045, USA.
Tel: (310) 338 0400.
URL: www.phoenixsoftware.com/EJES/ejes.html.

***

Mainstar Software has announced CATSCRUB, a new back-up and recovery manager suite selectable feature.

CATSCRUB synchronizes one or more BCS catalogs with associated DASD volumes at a disaster recovery site, resulting in catalogs that correctly reflect the data on the physical volumes. With the SIMULATE feature, users can determine in advance exactly what this command will do.

For further information contact:
Mainstar Software, PO 4132, Bellevue, WA 98009-4132, USA.
Tel: (425) 455 3589.
URL: www.mainstar.com.