157

MVS

update

October 1999

In this issue

MVS Update

Published by

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

North American office

Xephon/QNA
1301 West Highway 407, Suite 201-405
Lewisville, TX 75067
USA
Telephone: 940 455 7050

Contributions

If you have anything original to say about MVS, or any interesting experience to recount, why not spend an hour or two putting it on paper? The article need not be very long – two or three paragraphs could be sufficient. Not only will you be actively helping the free exchange of information, which benefits all MVS users, but you will also gain professional recognition for your expertise, and the expertise of your colleagues, as well as some material reward in the form of a publication fee – we pay at the rate of £170 ($250) per 1000 words for all original material published in *MVS Update*. If you would like to know a bit more before starting on an article, write to us at one of the above addresses, and we'll send you full details, without any obligation on your part.

Editor

Jaime Kaminski

Disclaimer

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

MVS Update on-line

Code from *MVS Update* can be downloaded from our Web site at http://www.xephon.com; you will need the user-id shown on your address label.

Subscriptions and back-issues

A year's subscription to *MVS Update*, comprising twelve monthly issues, costs £340.00 in the UK; $505.00 in the USA and Canada; £346.00 in Europe; £352.00 in Australasia and Japan; and £350.00 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 1992 issue, are available separately to subscribers for £29.00 ($43.00) each including postage.

Printed in England.

CA1 TMC information

During a recent change to our DASD layout it became necessary to move our CA1 TMC (Tape Management Catalog) to another volume. However, when we tried to restart CA1, we received an 'IEFTMS70 5yy-132' message. This message is issued in situations where either the TMC or the audit file has been incorrectly defined. Although I have worked with CA1 for quite a while, I was unable to understand what was wrong and had to resort to calling CA for assistance. The answer they provided was sufficiently unexpected that it seemed worthwhile pointing out the problem to a wider audience. Apparently the message was issued because CA1 could not handle 31-bit UCBs. We had hit the problem by defining our new DASD with 31-bit UCBs rather than our old DASD's 24-bit ones.

Having assisted various people trying to understand the operation of CA1 and its use of the TMC, there would appear to be a lot of confusion and concern over this particular file. A lot of this confusion seems to stem from the TMC being an immovable file that requires 'specialist' procedures to move it.

The TMC is defined as a 340-byte fixed record, unblocked, immovable file. If you were to treat it as such and run a utility like SELCOPY to print out the file it would consist of the following structure.

The first three records are control records responsible for defining the volume ranges in use at your site (up to 52 independent ranges) and for controlling DSNB usage. Following these control records will be one volume record for every volume defined in your volume range. At the end of these records and marked by X'FF' in the first byte are the DSNB records.

If you look in the macro library supplied by CA, you will find two macros – TMMTMREC and TMMDSNB. TMMTMREC defines the layout of the control records and the volume records, while TMMDSNB defines the DSNB layout. Reading these Assembler macros should enable you to be able to see a correlation between the GRW (Generalized Report Writer) and the records. In other words you should be able to see the TMC as nothing more than a strangely unblocked ordinary file. Using the macro layouts for information will also permit the use of routines other than the GRW (or EARL) to read the TMC. So if you lack the skills in either of these tools, you can simply use the maps as your guides for SELCOPY, etc. So if the TMC is just an ordinary file with an ordinary fixed format, why all the problem when it comes to moving it? There are two primary reasons:

- How the I/O is actually done to the TMC. When CA1 is started (or restarted) it takes note of where the TMC is. After this, access to the file is done through an SVC rather than through conventional I/O. This SVC works out the location of a TMC record and then 'jumps' to the disk location. This calculation of the offset from the start of the file is done by using the control records to work out where the currently in-use volume serial number is located and then adding in the number of control records. To make this clearer, assume you have two volume ranges, VOLSERs 10-19 and 30-49, and we are about to process volume 40. The jump is calculated as follows. Start of file +3records (for the control records) +10 records – VOLSER range 10-19 is 10 records – +11records (Volser 40-30 (start of second range) is 11 records). Hence, a jump to record 24 is required. As standard I/O is not done and is merely a relative record calculation, and because the start of file pointer is only obtained at start-up, should the file be moveable and, for example, defragged elsewhere, the I/O would continue to where the file was previously and not its new location.
- The TMC must not be in use while the move takes place. Any use of tape which could update the TMC while the move is happening would result in potential TMC damage. It is imperative therefore that tape activity is stopped while the move takes place.

Assuming you protect the TMC accordingly, you do not literaly need to follow the procedures as stated in the manual (not that I am recommending you do not follow them, of course). Utilities such as DFDSS are perfectly capable of safely moving a TMC as long as tape activity is controlled during the copy process and that CA1 is restarted after the move. It is the last point that matters most. You really can treat the TMC as an ordinary file and read and process it accordingly as long as CA1 is restarted to reset itself after the move and no tape activity has updated the TMC during your change activity.

Systems Programmer (UK) © Xephon 1999

COBOL II scope delimiters

INTRODUCTION

Have you ever needed to get into a compound ‘IF’ phrase and wished you could end one ‘IF’ so you could begin another – without using a full stop? This is where scope delimiters can be very helpful.

USING COBOL II SCOPE DELIMITERS

Consider this example of a Y2K fix that would cause serious problems if you were stuck with COBOL I and could not use scope delimiters. In the example, dates are in the format YYMMDD; if there is a ‘start’ date, there will be a corresponding ‘end’ date; if there is no ‘start’ date, there will be no ‘end’ date:

```
IF INPUT-START-DATE > ZERO
     MOVE INPUT-START-DATE        TO WS-START-DATE-6
     MOVE INPUT-END-DATE          TO WS-END-DATE-6
     IF WS-START-DATE-6 < 500000
         MOVE 20                  TO WS-START-DATE-CC
     ELSE
         MOVE 10                  TO WS-START-DATE-CC
```

Now we are in trouble. We want to repeat the last four lines of code for the end date without leaving the ‘IF’ sentence, since we only do all this if START-DATE > zero. We could repeat the ‘IF’ again, but that is wasteful. In COBOL II and above it can look like this:

```
IF INPUT-START-DATE > ZERO
     MOVE INPUT-START-DATE        TO WS-START-DATE-6
     MOVE INPUT-END-DATE          TO WS-END-DATE-6
     IF WS-START-DATE-6 < 500000
         MOVE 20                  TO WS-START-DATE-CC
     ELSE
         MOVE 10                  TO WS-START-DATE-CC
         END-IF
         IF WS-END-DATE-6 < 500000
              MOVE 20             TO WS-END-DATE-CC
         ELSE
              MOVE 10             TO WS-END-DATE-CC
         END-IF
     END-IF.
```

The last two ‘END-IF’s are superfluous, we could have put a full stop after the last ‘WS-END-DATE-CC’. Some programmers include them for style and readability.

‘END-IF’ terminates the scope of the immediately preceding ‘IF’ without terminating the sentence. Just like ‘ELSE’ matches up with the immediately preceding ‘IF’. Both require the same degree of care during use.

OTHER SCOPE DELIMITERS

There are a number of scope delimiters in COBOL II. A selection are shown below:

- END-ADD
- END-CALL
- END-DIVIDE
- END-EVALUATE
- END-MULTIPLY
- END-READ
- END-SUBTRACT
- END-WRITE.

We will consider some of these COBOL II scope delimiters in future issues of *MVS Update*.

Allan Kalar
Systems Programmer (USA)

JES2 recovery

THE PROBLEM

Recently an unusual situation occurred at our site which had not happened for many years. One of the datasets in a JES2 procedure concatenation had become corrupted. This meant that the batch jobs using that concatenation were failing to find their JCL. Although we were able to repair the dataset, it needed to be redefined and therefore JES2 was unaware of the new ‘correct’ library. In common with most sites, an IPL to restore matters was not exactly the preferred option! So it was time to try something that I had not used since the days of MVS/ESA Version 3 Release 3 (currently we are at OS/390 Version 1 Release 3). Because it worked, the procedure may be of use should you encounter the problem at your site.

A SOLUTION

The command £P JES2,ABEND was issued. This forces JES2 to stop and issue a variety of messages, ending with a WTOR. By replying END to the WTOR, and then issuing a S JES2,WARM,NOREQ it was possible to get JES2 to restart as a ‘hot’ start. This allowed JES2 to find its PROCLIB concatenation again, and (the best bit) nothing stopped or failed in the process.

Should you need to try this, an extract from the log to show all the messages obtained is shown below. This is followed by a description for what a ‘hot’ start ‘officially’ means in JES2 terms. I have included the log extract below, because the messages provided can look decidedly worrying if you are not ready for them!

```
£P JES2,ABEND
*£HASPØ95 JES2 CATASTROPHIC ERROR.  CODE = £PJ2
 £HASPØ88 JES2 ABEND ANALYSIS 842
 £HASPØ88 -----------------------
 £HASPØ88 FMID   = HJE66Ø3    LOAD MODULE = HASJES2Ø
 £HASPØ88 SUBSYS = JES2  OS 1.3.Ø
 £HASPØ88 DATE   = 99.217         TIME   = 19.27.11
 £HASPØ88 DESC = OPERATOR ISSUED £PJES2, ABEND
 £HASPØ88  MODULE   MODULE     OFFSET  SERVICE  ROUTINE
 £HASPØ88  NAME     BASE     + OF CALL LEVEL    CALLED
 £HASPØ88  ———— ————   ——— ———  —————
 £HASPØ88  HASPCOMM ØØØ7D828 + ØØEB56  OW21918 *ERROR £PJ2
 £HASPØ88 PCE  = COMM     (ØCA361BØ)
 £HASPØ88 RØ   = ØØØ8C15Ø  ØØBF4738  ØØØØ522E  ØØØ83F44
 £HASPØ88 R4   = ØØØ83Ø2C  ØCA36B2C  ØØØØØØØ4  ØCA36B3Ø
 £HASPØ88 R8   = ØØØ8C15Ø  ØØØ8D15Ø  ØØØØØØØØ  ØØØØ6ØØØ
 £HASPØ88 R12  = ØØØ7D878  ØCA361BØ  ØCAØ5CCØ  ØØØD37F8
 £HASPØ88 ———————————————————————
*£HASP198 REPLY TO £HASPØ98 WITH ONE OF THE FOLLOWING: 843
   END             - STANDARD ABNORMAL END
   END,NOHOTSTART - ABBREVIATED ABNORMAL END (HOT-START IS AT RISK)
   SNAP            - RE-DISPLAY £HASPØ88
   DUMP            - REQUEST SYSTEM DUMP (WITH AN OPTIONAL TITLE)
*71 £HASPØ98 ENTER TERMINATION OPTION

*72 £HASP426 SPECIFY OPTIONS - JES2 OS 1.3.Ø
 GSVXØ19I JES2 is not active
 R 72,WARM,NOREQ
 IEE6ØØI REPLY TO 72 IS;WARM,NOREQ
 IEF196I IEF285I   SYS1.PARMLIB                                 KEPT
 IEF196I IEF285I   VOL SER NOS= SY1ØA1.
 IXZØØØ1I CONNECTION TO JESXCF COMPONENT ESTABLISHED, 973
          GROUP MVSJESP1 MEMBER MVSJESP1£IPO1
 £HASP537 THE STATEMENTS IN THE INITIALIZATION DECK REQUIRE A 974
          CHECKPOINT SIZE OF 431 4K RECORDS
 IEF196I IEF237I 1Ø8A ALLOCATED TO SYSØØØØ1
 £HASP478 INITIAL CHECKPOINT READ IS FROM CKPT1 976
          (SYS1.HASPCKPT ON JESØØ1)
          LAST WRITTEN THURSDAY, 5 AUG 1999 AT 19:27:Ø8 (LOCAL TIME)
*£HASP493 JES2 MEMBER-IPO1 HOT START IS IN PROGRESS
£HASP492 JES2 MEMBER-IPO1 HOT START HAS COMPLETED
```

HOT START

The following is an extract from the JES2 commands manual that provides a definition of a hot start:

Hot start: a hot start is a warm start of an abnormally terminated JES2 member without an intervening IPL. JES2 performs a hot start when a particular JES2 member has stopped but other systems have continued to function and have not experienced problems. When JES2 hot starts, all address spaces continue to execute as if JES2 had never terminated. Jobs that were processing on output devices are re-queued as if a £I command had been issued. Jobs on input devices must be resubmitted and lines must be restarted. Hot starts have no effect on other members in a MAS configuration.

Converting a Julian date to Gregorian

INTRODUCTION

In *MVS Update*, Issue 156, September 1999, there was a simple utility to execute job steps based on the day of the week. The utility worked by extracting the system date in Julian format, converting this date into Gregorian format and then calculating the weekday using Zeller's Congruence.

However, there is no need to convert a Julian date to Gregorian format before finding the day of the week. The code below does it directly:

```
DAYOWEEK CSECT
         SAVE  (14,2),,*
         LR    R2,R15
         USING DAYOWEEK,R2
         LA    R14,SAVEAREA
         ST    R13,4(,R14)
         ST    R14,8(,R13)
         LR    R13,R14
         TIME  LINKAGE=SVC         GET CURRENT DATE
         ST    R1,WORK1+4          STORE DATE IN ØCYYDDDF FORMAT
         DP    WORK1,=PL3'1ØØØ'    SPLIT YEAR AND DAY
         ZAP   WORK2,WORK1(5)      PICK UP YEAR
         CVB   R15,WORK2           YEAR IN BINARY
         M     R14,=F'5'           MULTIPLY BY 5
         BCTR  R15,Ø               SUBTRACT 1
         SRA   R15,2               DIVIDE BY 4
         ZAP   WORK2,WORK1+5(3)    PICK UP DAY
         CVB   R14,WORK2           DAY IN BINARY
         AR    R15,R14             ADD TO "YEAR"
         SR    R14,R14             ZEROISE EVEN REGISTER
         D     R14,=F'7'           GET DAY OF WEEK
         LR    R15,R14             PUT RESULT IN REGISTER 15
         L     R13,4(,R13)         RESTORE REGISTER 13
         RETURN (14,2),RC=(15)     Ø=SUN, 1=MON, 2=TUE, ... 6=SAT
*
SAVEAREA DC    9D'Ø'
WORK1    DC    D'Ø'                WORK AREA
WORK2    DS    D                   WORK AREA
         LTORG
         YREGS
         END   DAYOWEEK
```

Dave Thorby
Systems Programmer
London Electricity plc (UK)

PDF line commands

INTRODUCTION

In edit mode, you can enter line commands in the left margin (the line number area). You probably already know the most common ones such as ‘C’opy, ‘A’fter, ‘B’efore, ‘M’ove, ‘D’elete, ‘I’nsert, ‘R’epeat, etc, and their block equivalents. There are others such as: ‘O’verlay, ‘)’ shift right, ‘(‘shift left, as well as ‘TS’ text split, and ‘TF’ text flow.

LINE COMMANDS

Here are a few of the less common tools available:

- ‘X’ will temporarily hide (or e‘X’clude) a line. ‘XX’ is the group equivalent. This is handy to get some intervening lines out of the way so that you can get two groups of lines closer together, perhaps on the same screen. The line or lines are replaced with a single line of dashes. You can enter the following commands in the left margin of that line (If ‘n’ is not specified, it automatically defaults to ‘1’).

- ‘Fn’ will redisplay the first ‘n’ lines of excluded text.

- ‘Ln’ redisplays the last ‘n’ lines.

- ‘Sn’ redisplay ‘n’ lines with the leftmost indentation in a block of excluded lines.

- ‘UC’ will convert all the characters on a line to upper case. ‘UCC’ is the group command. ‘LC’ and ‘LCC’ are the ‘lower-case’ equivalents.

Information about the features that involve a ‘mask’ can be found in the information in HELP (PF1).

Allan Kalar
Systems Programmer (USA)

OS/390 Version 2 Release 8

INTRODUCTION

24 September 1999 saw the general availability of OS/390 Version 2 Release 8. The emphasis of the new release is on better integration and availability. Central to the upgrade are security and systems management features, including the ability to manage virtual private network encryption keys dynamically through the Internet Key Exchange, enhanced management and administration of digital certificates used by both server applications and end-users, higher availability of TCP/IP in a Parallel Sysplex, ISPF customization, and the capability to print from ERP and Internet-related applications.

NEW SECURITY FEATURES

The following new security features are particularly important and will enhance the role of OS/390 in supporting e-commerce:

- IPSec VPN provides a secure pathway between OS/390 and other IPSec VPN-capable systems, routers, and firewalls through encryption using the System/390 hardware CMOS Cryptographic Coprocessor.

- The exchange of encryption keys between the end-points of IPSec VPN can now be automated and dynamically managed through Internet Key Exchange (IKE), an IPSec protocol for cryptographic key and security management.

- The centralized management of digital certificates belonging to server applications and their related private encryption keys, is another new feature. This will allow customers and application developers to provide common secure management of these certificates as well as the chain of trust needed to verify user certificates presented to these applications.

- There is SSL client authentication to the TN3270 server, allowing TCP/IP clients to access customer applications traditionally only accessible from a 3270 screen. It also secures against unauthorized access to SNA applications from TCP/IP users.

LDAP DIRECECTORY ACCESS

The LDAP (Lightweight Directory Access Protocol) server has been enhanced to support LDAP Version 3 protocol, enabling OS/390 LDAP Server to interoperate with other LDAP Version 3 clients and servers. LDAP on OS/390 includes Java support, LDAP access to RACF information, and LDAP client authentication using RACF. It also supports the Secure Sockets Layer for encrypted privacy of communication and it supports multiple LDAP servers on multiple systems in a Parallel Sysplex.

VIRTUAL IP ADDRESSING

The SecureWay Communications Server for OS/390 provides Virtual IP Addressing (VIPA) Takeover, allowing real IP addresses to be associated with a pseudo address, assigned to an end-user in the System/390 server. If a connection fails, traffic is routed to an alternative connection associated with the same VIPA.

PRINTER CONSOLIDATION

Another new feature is that Infoprint Server will use the Internet printing protocol to process print jobs over the Internet securely. Infoprint will use datastream transforms to translate data from one printer format to another to allow printing from popular PC and workstation applications as well as many ERP applications. It can convert PCL, Postscript, and PDF files and is being positioned to consolidate enterprise print serving functions around the System/390.

PARALLEL SYSPLEX CONFIGURATION

Parallel Sysplex configuration can be complex but the use of the Parallel Sysplex configuration tool allows a Parallel Sysplex configuration to be created interactively.

ISPF CUSTOMIZATION

One of the problems of installing new releases of OS/390 has been that ISPF customization needs to be redone every time. Normally this involves using the ISRCONFG Assembler resource, and manually reassembling and relinking to make changes. With the introduction of the ‘ISPF configuration utility’ this problem is alleviated. Upon entering TSO ISPCCONF, it is possible to convert the old ISRCONFG Assembler source into a keyword file that includes all the options or all those changed from the defaults.

OTHER ENHANCEMENTS

Other enhancements to OS/390 Version 2 Release 8 include:

- Allowing SNA users to use Triple DES encryption. Service policy enhancements are said to improve the capability to monitor and manage network performance to service level agreements.
- The addition of a dynamic update feature to the policy agent allows service policies to be implemented without impacting network availability.
- A new resource reservation protocol support allows users to invoke reservation services, reserve bandwidth, and classify reservations through an RSVP API.
- A new SNMP SLA subagent that enables network administrators to retrieve data and determine whether the current set of SLA policy definitions are performing as desired.
- Improvements in Workload Manager (WLM) that allow it to prioritize workloads at the request level.

ANALYSIS

As with other recent releases of OS/390, the emphasis on security implies the operating system is being aggressively promoted for electronic commerce applications. The emphasis on printer consolidation, and Parallel Sysplex configuration again re-emphasizes the importance of enterprise server centralization.

OS/390 VERSION 2 RELEASE 9

In Version 2 Release 9, IBM plans to make further improvements to native file and print serving for Windows clients, text search support for XML documents and unicode, and additional Unix system services functions. We can also look forward to a new naming convention for OS/390 in the new millennium.

Selectively blocking commands

THE PROBLEM

It is sometimes necessary to block the issue of certain MVS commands. You may for instance have a job that you do not want CANCELLed or even FORCEed without permission from the systems programming group. There are also commands that can have an impact on system performance (eg doing an LLA refresh on a production system). Although this can be controlled through RACF profiles, it is a somewhat cumbersome process and a mistake could easily end up with the wrong groups being blocked or allowed to issue the command(s).

A SOLUTION

A much easier way would be to have a simple facility available that will allow systems programmers to block commands by putting them into and removing them from a PARMLIB member.

The following facility does exactly that. There are two programs – an MVS command exit, and a routine to manage blocked commands. The MVS command exit intercepts commands of the format 'T NOCMD=xx' with xx the suffix of a PARMLIB member NOCMDxx.

This command is passed to the second routine that manages the blocked commands. This routine looks for the member NOCMDxx in the PARMLIB concatenation and, if the member is found, a CSA table is GETMAINed and the commands are copied from the member into the table. The address of the CSA table is anchored off a subsystem entry by the name of NCMD. (This entry is dynamically added when the first 'T NOCMD=xx' command is entered.) The command exit does a check every time that a command is entered. If the name of the subsystem exists, the command is compared with all the entries in the CSA table. If a match is found, the command is not executed and a message is given. To disable the command blocking, simply issue a 'T NOCMD=NO' command (which of course can also be put into the block table).

Note that the commands are added into the PARMLIB member without blanks, eg to block a LLA refresh the PARMLIB member should have ‘FLLA’, on a line of its own. The comparison with the entered command is based on the length of the command in the PARMLIB member. For example, CANCELABC will block the command ‘CANCEL ABCD’ but it will not block the command ‘CANCEL A’. The program is not written to accept wild cards (eg CANCEL AB*D) but it can easily be modified to accept this format.

The MVS command exit also contains samples on how to intercept other commands and will be very easy to convert to suit your purposes. It removes the blanks from the passed commands and then calls the associated entry points for each matched command. The field CMDWAREA contains the de-blanked command whilst CMDX@ contains the address of the original MVS command buffer. It is installed by adding it into MPF refer to the *OS/390 Initialization Guide* for a description on how to do this very easy task.

MVS COMMAND EXIT

```
************************************************************************
* MVS Command EXIT.
* This module will recognize commands by comparing them to entries in
* a table. It will than do a BAS to an entry point specified for the
* command. If the processing is successful, it will either suppress the
* command from further MVS processing or pass it to MVS, depending on
* a field specified per command in the command table.
* An ESTAE routine is set up to intercept any ABENDs that may result in
* calling the subroutines. The ESTAE routine issues an SVC dump.
* To add another command intercept:
* 1) Do a "F COMMTAB" in column 1. This is the command table and an
*    entry can be added The format is:
*        DC   AL2(L'XX)            .Length of command text
*        DC   AL4(ENTRYPT)         .Address of processing routine
*        DC   AL1(flag)            .Flag to control MVS processing
*   XX   DC   C'MYCOMMND'          .Command text (spaces removed)
* 2) The code to process command MYCOMMND should be coded at ENTRYPT.
* 3) Assemble and link.
* 4) Refresh LLA
* 5) Do a "T MPF=xx"
* Module type    :   Reentrant, called in supervisor state key Ø
* Addressing mode:   AMODE 31, RMODE any
* Register usage :   R12= Base register
*                    R13= Pointer to general savearea and workareas
***********************************************************************
MVSCOMEX CSECT
MVSCOMEX AMODE 31
MVSCOMEX RMODE ANY
         BAKR  R14,Ø                .Save caller's status
         LR    R12,R15
         USING MVSCOMEX,R12
**********************************************************************
*        Main driver routine
**********************************************************************
         L     R4,Ø(R1)             .Get CMDX Address
         USING CMDX,R4
STORAGE  LA    R3,STORSIZE
         STORAGE OBTAIN,LENGTH=(3),LOC=ANY,SP=229
         LA    R3,STORSIZE
         LR    R2,R1                .Point to getmained area
         XR    R9,R9
         MVCL  R2,R8                .Propagate binary zeros
         USING STORAREA,R1
         ST    R13,SAVEAREA+4       .Backchain
         DROP  R1
         LR    R13,R1
         USING STORAREA,R13         .Addressability to getmained area
         ST    R4,CMDX@             .Preserve adress of command buffer
         BAS   R14,FINDCMND         .Locate command text, remove blanks
CHKPRCS  BAS   R14,CMNDCOMP         .See if we have to process command
         BAS   R14,SEEKSSI          .Go see if command blocking active
         LTR   R15,R15              .Is it?
         BZ    CHKOURS              .No, go see if we have to intercept
         BAS   R14,TSTBLOCK         .Compare command with "block" table
         LTR   R15,R15              .Must it be blocked?
         BZ    CHKOURS              .No, don't block the command
         WTO   'MVSCOMEX(W): This command has been blocked by Software X
               Support',ROUTCDE=11
CHKOURS  TM    DOFLAG,YES           .Is this an intercepted command?
         BNO   RETURN               .No, not one of ours
         BAS   R14,PROCCMND         .Process command
RETURN   L     R4,RETCODE           .Pick up return code
         LA    R3,STORSIZE
         LR    R2,R13               .Pointer to storage area
         STORAGE RELEASE,LENGTH=(3),SP=229,ADDR=(2)
         LR    R15,R4               .Reload return code
         PR                         .Return to our caller
**********************************************************************
*        This routine locates the command and removes all blanks
**********************************************************************
FINDCMND BAKR  R14,Ø
         L     R4,CMDXCLIP          .Get command buffer address
         DROP  R4
         USING CMDXCLIB,R4          .Access the passed command buffer
         LA    R1,CMDXCMDL          .MVS buffer of actual command text
         ST    R1,MVSCOM@           .Preserve this address
         LH    R1,CMDXCMDL          .Length of command
         LA    R2,CMDXCMDI          .Access start of text
POINTCMD LA    R3,CMDWAREA          .Point to start of command wrk area
```

```
         XR    R5,R5                .Clear length counter
DESPACEL EQU   *                    .Remove all spaces from the command
         CLI   Ø(R2),C' '           .Is it a space?
         BE    BUMPUP               .Yes
         CLI   Ø(R2),X'ØØ'          .Is it low value?
         BE    BUMPUP               .Yes
         CLM   R5,3,=AL2(L'CMDWAREA) Too long?
         BL    MOVECHAR             .No
         WTO   'MVSCOMEX(E): -Command exceeds max length',ROUTCDE=13
         B     UPPERCSE             .Only accept these characters
MOVECHAR MVC   Ø(1,R3),Ø(R2)        .Move character into wrk area
         LA    R3,1(R3)             .Point to next empty space
         LA    R5,1(R5)             .Bump up length counter by 1
BUMPUP   LA    R2,1(R2)             .Bump up pointer to cmd buffer
         BCT   R1,DESPACEL          .Do for each character in the cmd
UPPERCSE OC    CMDWAREA(L'CMDWAREA),SPACES .Uppercase
         ST    R5,CMDLENG           .Length of de-spaced command
FINDCMNX PR                         .Reload our return address
*********************************************************************
*        This routine gets the address of our subsystem entry
*********************************************************************
SEEKSSI  BAKR  R14,Ø
         L     R4,16                .Start of CVT
         USING CVT,R4               .Establish addressability
         L     R4,CVTJESCT          .Get JESCT address
         USING JESCT,R4             .Establish addressability
         L     R4,JESSSCT           .Get SSCT chain address
         USING SSCT,R4              .Set addressability
SSCTLOOP EQU   *                    .Look for our SSCVT
         LTR   R4,R4                .End of chain?
         BZ    NOBLOCK              .Yes - not found
         CLC   SSCTSNAM,SUBSYSN     .Our SSCVT?
         BE    FOUNDSSI             .Gotcha
         L     R4,SSCTSCTA          .Point to next SSCVT entry
         B     SSCTLOOP             .Redo the loop
FOUNDSSI EQU   *                    .We have found the subsystem
         CLC   SSCTSUSE,=F'Ø'       .Has the CSA table been obtained?
         BZ    NOBLOCK              .No, we are not blocking commands
         L     R15,SSCTSUSE         .Return address of the CSA table
         ST    R15,BLOCKTB@         .Store the address
         B     SEEKSSIX             .Get out
NOBLOCK  XR    R15,R15              .No, we are not blocking commands
SEEKSSIX PR
***********************************************************************
*        This routine checks if the command matches an entry in the
*        "block" table kept in CSA.
***********************************************************************
TSTBLOCK BAKR  R14,Ø
         L     R7,BLOCKTB@          .Where the CSA "block" table is
         L     R6,Ø(R7)             .Number of entries in "block" table
         LTR   R6,R6                .Any commands in the table?
         BZ    ALLOWCMD             .No commands in "block" table
         LA    R7,4(R7)             .Start of first command text
CARDLOOP EQU   *
         LA    R9,8Ø                .Length of a "block" cmd entry
         LR    R1,R7                .Point to start of "block" cmd
         XR    R8,R8                .Clear length counter
LENGLOOP CLI   Ø(R1),X'4Ø'          .Look for a space
         BE    SETLENG              .Command must match this length
         LA    R1,1(R1)             .Point to next character
         LA    R8,1(R8)             .Bump up length counter by 1
         BCT   R9,LENGLOOP          .Scan until we find a blank or EOC
SETLENG  L     R3,CMDLENG           .Length of de-spaced command
         C     R3,R8                .Command buffer too short?
         BL    ALLOWCMD             .Yes, do not block the command
         LR    R5,R8                .Length to compare
         LR    R3,R8                .Length to compare
         LR    R2,R7                .Point to start of this "block" cmd
         LA    R4,CMDWAREA          .Point to start of passed command
         CLCL  R2,R4                .Does command match a table entry?
         BE    BLOCKCMD             .Yes, block the command
         LA    R7,8Ø(R7)            .Point to the next card
         BCT   R6,CARDLOOP          .Compare with each table entry
ALLOWCMD XR    R15,R15              .This command not blocked
         B     TSTBLOCX             .Get out
BLOCKCMD LA    R15,NOMVS            .Do not let this through to MVS
         ST    R15,RETCODE          .Plug value into return code field
TSTBLOCX PR
***********************************************************************
*        This routine compares the command to entries in the table
***********************************************************************
CMNDCOMP BAKR  R14,Ø
         LA    R4,COMMTAB           .Point to start of command table
         DROP  R4
         USING COMMDSCT,R4          .Addressability to command table
         LA    R3,ENDOFTAB          .Point to end of command table
COMPLOOP EQU   *                    .Compare to all commands in table
         ICM   R2,3,COMMLENG        .Pick up command length
         STH   R2,CMDTXTLN          .Store into parmlist
         BCTR  R2,Ø                 .Reduce length by 1
         EX    R2,COMCOMPR          .See if there is a match
         BE    MATCHFND             .Yes, there is
         B     NOMATCH
COMCOMPR CLC   COMMTEXT(Ø),CMDWAREA .Compare command to table entry
NOMATCH  XR    R1,R1
         ICM   R1,3,COMMLENG        .Pick up the length of this entry
         AR    R4,R1                .Bump up by the text length
         LA    R4,COMMHDR(R4)       .Bump up by header length
         CR    R4,R3                .End of table?
         BNL   CMNDCOMX             .Yes, match not found. Get out
         B     COMPLOOP             .Redo for each table entry
MATCHFND EQU   *
         ST    R4,MATCH@            .Address of matched table entry
         WTL   'MVSCOMEX(I): -Command analysed by exit'
         OI    DOFLAG,YES           .Yes, it must be processed by us
CMNDCOMX PR                         .Return to our caller
************************************************************************
*        This routine sets up the ESTAE and calls the correct EP
************************************************************************
PROCCMND BAKR  R14,Ø
         LA    R14,PROCCMNX         .Start from here after an abend
         STM   R1,R14,RUBLSRGS      .Preserve all our registers
         MVC   RUBLIST,=B'Ø11111111111111Ø'  Regs 1-14 to be reloaded
         LA    R1,ESTAESTR          .Point to ESTAE storage area
         MVC   Ø(ESTAEL1,R1),ESTAEMAC
         ST    R13,ESTAEPRM         .R13 required by recovery routine
         LA    R2,ESTAEPRM          .Pass R13 contents as a parameter
         ESTAE MF=(E,(1)),PARAM=((2))
         L     R4,MATCH@            .Address of matching cmd tble ent
         ICM   R15,15,COMMEP        .Entry point of routine to call
         BASR  R14,R15              .Call command processing routine
         CLC   RETCODE,=F'Ø'        .Was the processing successful?
         BNZ   PROCCMNX             .No, pass the command to MVS
         XR    R15,R15
         IC    R15,COMMFLAG         .What MVS should do when we're done
         CH    R15,=AL2(TODECIDE)   .Dynamic decision on what to do?
         BNE   PROCCMNX             .No, do as specified in table
         L     R15,DECISION         .Pick up our decision
PROCCMNX ST    R15,RETCODE          .This is what we pass back to MVS
         LA    R1,ESTAESTR          .Point to ESTAE storage area
         MVC   Ø(ESTAEL2,R1),ESTAEDEL
         ESTAE MF=(E,(1))           .Remove ESTAE
         PR                         .Return to our caller
**********************************************************************
*        EXAMPLE: This routine processes the "D TS,L" command
**********************************************************************
ROUTINEA BAKR R14,Ø
         WTL  '"D TS,L" Command detected'
*  Add code in here to do processing for "D TS,L"...
         ST   R15,RetCode
         PR
**********************************************************************
*        EXAMPLE: This routine processes the "CANCEL" command
**********************************************************************
ROUTINEB BAKR R14,Ø
         WTL  '"CANCEL" Command detected'
         ST   R15,RetCode
*  Add code in here to do processing for "CANCEL"...
*  Set field DECISION depending on whether the command is acceptable
*  and should also be passed to MVS.
         PR
***********************************************************************
*        This routine handles the "T NOCMD=' command
***********************************************************************
TNOCMD   EQU   *
         BAKR  R14,Ø                .Preserve our registers
         LA    R2,CMDWAREA          .Point to (compressed) command
         ST    R2,CALLPRM           .Plug the address into parmarea
         LA    R1,CALLPRM           .Address of call parms
         ST    R1,CALLPRM@          .Plug into pointer
         LA    R1,CALLPRM@          .Point to parameter area
         LINK  EP=TELNOCMD          .Call the routine to do the work
TNOCMDX  ST    R15,RETCODE
         PR
*********************************************************************
*        Constants follow
**********************************************************************
COMMTAB  DS   ØF                    .Add new commands in here
*********************************************************************
COMADESC DC   AL2(L'VERBA)          .Length of command text
         DC   AL4(ROUTINEA)         .Address of processing routine
         DC   AL1(TOMVS)            .Flag: pass command to MVS
VERBA    DC   C'DTS,L'              .Command text without blanks
*********************************************************************
COMBDESC DC   AL2(L'VERBB)          .Length of command text
         DC   AL4(ROUTINEB)         .Address of processing routine
         DC   AL1(TODECIDE)         .Flag: do not pass command to MVS
VERBB    DC   C'CANCEL'             .Command text without blanks
*********************************************************************
COMCDESC DC   AL2(L'VERBC)          .Length of command text
         DC   AL4(TNOCMD)           .Address of processing routine
         DC   AL1(NOMVS)            .Flag: do not pass command to MVS
VERBC    DC   C'TNOCMD='            .Command text without blanks
*********************************************************************
*  If the called entry point for a specific command returns a non-zero
*  return code, the command is always also passed to MVS, as we failed.
*  If the entry point returns a zero return code (in R15), a decision
*  as to what should be done is based on the third field for the entry
*  in the table, which must always be one of the following.
*  "TODECIDE", means that whether to pass the command to MVS
*  or not is to be decided. A further field called "Decision" is then
*  scanned. If this field contains the value ØØ, the command is also
ENDOFTAB EQU  *
TOMVS    EQU   X'ØØ'                .Pass the command to MVS as well
NOMVS    EQU   X'Ø4'                .Do not pass the command to MVS
TODECIDE EQU   X'Ø1'                .Decision to be made on passing
**********************************************************************
COMMTABL EQU  *-COMMTAB
SPACES   DC   CL(L'CMDWAREA)' '
ESTAEMAC ESTAE RECOVER,PARAM=ESTAEMAC,ASYNCH=NO,MF=L
ESTAEL1  EQU   *-ESTAEMAC
ESTAEDEL ESTAE Ø,MF=L
ESTAEL2  EQU   *-ESTAEDEL
SUBSYSN  DC    CL4'NCMD'            .Name of subsystem with address of
         LTORG
***********************************************************************
*        This routine is the ESTAE error recovery routine
***********************************************************************
         DS    ØF
RECOVER  EQU   *
```

```
         LR    R12,R15              .Load our current address
         DROP  R12
         USING RECOVER,R12
         CH    RØ,=H'12'            .Make sure SDWAREA is available
         BNE   SDWAVAIL
NOTAVAIL XR    R15,R15              .SDWA is not available
         BR    R14                  .Go Back to MVS (percolate)
SDWAVAIL EQU   *                    .Remember SDWA address
         USING SDWA,R1
         LR    R3,R1                .Preserve R1
         L     R1,SDWAPARM          .Address of passed parm
         L     R13,Ø(1)             .Reload pointer to workarea
         ST    RØ,SDWASTOR          .Pointer to SDWA
         ST    R14,ESTAER14         .Our return address
         MVC   SDMPAREA(SDUMPLEN),SDUMPMAC
         LA    R1,SDMPAREA
         SDUMP MF=(E,(1))
         L     R2,RUBLSRGS+52       .Retry address
         LR    R1,R3                .Restore register 1
         LA    R3,RUBLIST
         SETRP RC=4,RETADDR=(2),RETREGS=YES,RUB=(3),DUMP=YES
         L     R14,ESTAER14         .Reload our return address
         BR    R14                  .Back to MVS
SDUMPMAC SDUMPX HDR='MVSCOMEX Command exit routine ABEND',             x
               SDATA=(RGN,SUM),MF=L
SDUMPLEN EQU   *-SDUMPMAC
         LTORG
*********************************************************************
*        DSECTs follow
*********************************************************************
STORAREA DSECT
SAVEAREA DS    18F                  .General savearea
CMDWAREA DS    CL2ØØ                .Command moved in here
CMDTXTLN DS    H                    .Length of (compressed) command
RETCODE  DS    F                    .SUBTASK's TCB
SDMPAREA DS    CL(SDUMPLEN)         .SDUMP macro area
BLOCKTB@ DS    F                    .Address of NCMD subsystem
DECISION DS    F                    .(dynamic) ret code to pass to MVS
CMDLENG  DS    F                    .Length of command without blanks
MATCH@   DS    F                    .Address of matching cmdtble entry
RUBSTART DS    ØF
         DS    H
RUBLIST  DS    H
RUBLSRGS DS    15F                  .Local registers used by ESTAE
ESTAEPRM DS    F                    .Parameter passed to ESTAE routine
SDWASTOR DS    F                    .Address of passed SDWA
ESTAER14 DS    F                    .Back-to-MVS address for ESTAE
ESTAESTR DS    CL(ESTAEL1)          .ESTAE macro area
*  The folowing two fields are passed to the called subroutine.
CALLPRM@ DS    F                    .Address of routine specific parms
CMDX@    DS    F                    .Address of CMDX as received
*
```

```
MVSCOM@  DS    F                    .Address (in MVS) of command text
DOFLAG   DS    C                    .Do we process command?
YES      EQU   X'8Ø'
NO       EQU   X'ØØ'
CALLPRM  DS    ØF
STORSIZE EQU   *-STORAREA
COMMDSCT DSECT                      .Command table DSECT
COMMLENG DS    CL2                  .Length of text to scan for
COMMEP   DS    AL4                  ."entrypoint" to branch to
COMMFLAG DS    C                    .MVS to process/ ignore instruction
COMMHDR  EQU   *-COMMDSCT           .Length of fixed part
COMMTEXT DS    ØC                   .Variable length text to scan for
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
         IEZVX1Ø1
         IHASDWA
         CVT DSECT=YES
         IEFJESCT
         IEFJSCVT
         END
```

BLOCKCMD SOURCE

This routine blocks certain commands from being entered.

```
BLOCKCMD CSECT
BLOCKCMD AMODE 31
BLOCKCMD RMODE 24
         BAKR  R14,Ø               .Save caller's status
         LR    R12,R15
         USING BLOCKCMD,12
**********************************************************************
*        Main driver routine
**********************************************************************
LOAD     L     R4,Ø(R1)            .Ptr to compressed command text addr
         L     R4,Ø(R4)            .Point to compressed command text
         L     R5,4(R1)            .Ptr to command exit info area
         USING CMDX,R5             .Addressability to command info
         LA    R3,STORSIZE         .Our requirement
         A     R3,BUFFSIZE         .Add length of the PARMLIB buffer
STORAGE  STORAGE OBTAIN,SP=229,LENGTH=(3),LOC=BELOW,KEY=8
         LR    R2,R1               .Point to getmained area
         LA    R3,STORSIZE
         XR    R9,R9
         MVCL  R2,R8               .Propagate binary zeroes
         USING STORAREA,R1
         ST    R13,SAVEAREA+4      .Back chain
         DROP  R1
         LR    R13,R1
         USING STORAREA,R13        .Addressability to getmained area
         MVC   CART,CMDXCART       .Pick up the CART
         MVC   FROMCNID,CMDXC4ID   .Where the command originated (Id)
         MVC   FROMSYS,CMDXISYN    .Where the command originated (Sys)
         L     R5,CMDXCLIP         .Get command buffer address
         DROP  R5
         USING CMDXCLIB,R5         .Access the passed command buffer
GOGETSFX BAS   R14,GETSUFIX        .Locate the data set name in the text
         LTR   R15,R15             .Did we get the suffix?
         BNZ   RETURN              .No, get out
         CLC   SUFFIX,=C'NO'       .Must we turn blocking off?
         BNE   GOREADIT            .No
         B     GOGETSSI            .Go get SSI address
GOREADIT BAS   R14,READPMEM        .Go read the PARMLIB member
         LTR   R15,R15             .Did we get the suffix?
         BNZ   RETURN              .No, get out
GOGETSSI BAS   R14,GETSSI@         .Go get the subsystem address
         LTR   R15,R15             .Did we get the address?
         BNZ   EXITMSGS            .No, get out
GOGETCSA BAS   R14,GETCSA@         .Go get the CSA table address
         LTR   R15,R15             .Did we get the address?
         BNZ   EXITMSGS            .No, get out
         BAS   R14,COPYCRDS        .Go copy cards into CSA table
EXITMSGS CLC   SUFFIX,=C'NO'       .Did we turn blocking off?
         BNE   CHEMPTY1            .No
         L     R8,FROMCNID         .Where command came from
         LA    R9,FROMSYS          .Where command came from
         WTO   'BLOCKCMD(I): -Command blocking has been turned off',   X
               CONSID=(8),SYSNAME=(9),CART=CART,ROUTCDE=11
         B     RETURN
CHEMPTY1 TM    EMPTY,YES           .Is the PARMLIB member empty?
         BO    RETURN              .Yes, get out
         L     R8,FROMCNID         .Where command came from
         LA    R9,FROMSYS          .Where command came from
         WTO   'BLOCKCMD(I): -Command blocking has been activated',    X
               CONSID=(8),SYSNAME=(9),CART=CART,ROUTCDE=11
RETURN   EQU   *                   .Pick up return code
         L     R4,RETCODE          .Pick up return code
         LA    R3,STORSIZE         .Size of area to free
         A     R3,BUFFSIZE         .Add length of the PARMLIB buffer
         LR    R2,R13              .Address of area to free
         STORAGE RELEASE,LENGTH=(R3),ADDR=(R2),SP=229,KEY=8
         LR    R15,R4              .Copy return code
TOCALLER PR    ,                   .=>Caller
         DS    ØD                  .Align
         EJECT
**********************************************************************
*        This routine locates the member suffix in the text
**********************************************************************
GETSUFIX BAKR  R14,Ø
         USING CMDXCLIP,CMDXCLIB   .Map to command length and text
         CLC   CMDXCMDL,=H'9'      .Length must be at least 9 bytes
         BL    INVLSNTX            .Invalid syntax
         CLI   9(R4),X'4Ø'         .Must have a blank in col 9
         BNE   INVLSNTX            ...of deblanked command buffer
         CLI   8(R4),X'4Ø'         .Must not have a blank in col 8
         BE    INVLSNTX            ...of deblanked command buffer
         CLI   7(R4),X'4Ø'         .Must not have a blank in col 7
         BNE   SYNTAXOK            ...of deblanked command buffer
INVLSNTX L     R8,FROMCNID         .Where command came from
         LA    R9,FROMSYS          .Where command came from
         WTO   'BLOCKCMD(E): -Invalid command syntax, must be "T NOCMD=X
               XX',CONSID=(8),SYSNAME=(9),CART=CART
         LA    R15,8
         ST    R15,RETCODE         .Set non-zero return code
         B     GETSUFXX            .Get out
SYNTAXOK MVC   MEMBER,=C'NOCMD'    .First part of the
         MVC   SUFFIX,7(R4)        .Pick up the suffix from cmd buffer
         MVI   LASTBYTE,X'4Ø'      .Move a blank into the last byte
         XR    R15,R15             .Clear return code
GETSUFXX PR
**********************************************************************
*        This routine locates the member in the PARMLIB concatenation
**********************************************************************
READPMEM BAKR  R14,Ø
         NI    EMPTY,NO            .Make sure 'PARMLIB empty' flag off
         LA    R2,WORKBUFF         .Where we want the member content
         USING PRM_READ_BUFFER,R2  .Addressability to buffer
         L     R1,BUFFSIZE         .Total size of the buffer
         ST    R1,PRM_READ_BUFF_SIZE
         LA    R1,IEFPRMLA         .Point to the macro in storage
         MVC   IEFPRMLA(IEFPRMLL),IEFMACRO
         IEFPRMLB  REQUEST=ALLOCATE,CALLERNAME=CALLER,READ=YES,        X
               READBUF=(R2),MEMNAME=MEMNAME,MF=(E,(1)),                X
               ALLOCDDNAME=PARMDDØ1,RETCODE=RETCODE,RSNCODE=RSNCODE
         LTR   R15,R15             .Was the member read in?
         BZ    FREEPARM            .Yes, go free it
READERR  L     R8,FROMCNID         .Where command came from
         LA    R9,FROMSYS          .Where command came from
         MVC   ALOCWTOA(ALOCWTOL),ALOCWTOM
         BAS   R14,CODEPRNT        .Make RC and REASON printable
         MVC   ALOCWTOA+72(4),PRTRC
         MVC   ALOCWTOA+87(4),PRTRSN
         LA    R1,ALOCWTOA
         WTO   MF=(E,(1)),CONSID=(8),SYSNAME=(9),CART=CART
         LA    R15,8
         ST    R15,RETCODE         .Set non-zero return code
         B    READPMEX             .Get out
FREEPARM IEFPRMLB  REQUEST=FREE,CALLERNAME=CALLER,                     X
               DDNAME=PARMDDØ1,RETCODE=RETCODE,RSNCODE=RSNCODE
         LTR   R15,R15             .Was the de-allocation successful?
         BNZ   FREEERR             .No.
CHKNUM   CLC   WORKBUFF+4(4),=F'Ø' .Did we get an empty PARMLIB member?
         BNE   READPMEX            .No
NOCARDS  L     R8,FROMCNID         .Where command came from
         LA    R9,FROMSYS          .Where command came from
         WTO   'BLOCKCMD(W): -No commands will be blocked as PARMLIB meX
               mber is empty',CONSID=(8),SYSNAME=(9),CART=CART
         OI    EMPTY,YES           .Se 'empty' flag
         XR    R15,R15             .Proceeding with empty table
         ST    R15,RETCODE         .Set non-zero return code
         B     READPMEX            .Get out
FREEERR  L     R8,FROMCNID         .Where command came from
         LA    R9,FROMSYS          .Where command came from
         MVC   FREEWTOA(FREEWTOL),FREEWTOM
         BAS   R14,CODEPRNT        .Make RC and REASON printable
         MVC   FREEWTOA+58(4),PRTRC
         MVC   FREEWTOA+73(4),PRTRSN
         LA    R1,FREEWTOA
         WTO   MF=(E,(1)),CONSID=(8),SYSNAME=(9),CART=CART
         LA    R15,8
         ST    R15,RETCODE         .Set non-zero return code
READPMEX PR
*********************************************************************
*        This routine returns the address of our subsystem.
*********************************************************************
GETSSI@  BAKR  R14,Ø
         BAS   R14,SEEKSSI         .Get our subsystem's address
         LTR   R15,R15             .Did we get the subsystem name?
         BZ    GETSSI@X            .Yes, get out
         CLC   SUFFIX,=C'NO'       .Are we turning blocking off?
         BNE   CHKEMPT1            .No, go see if member is empty
         LA    R15,8               .Yes, we don't need the SSI address
         B     GETSSI@X            .Get out
CHKEMPT1 TM    EMPTY,YES           .Is PARMLIB member empty anyway?
         BNO   ADSUBSYS            .No, there are entries in it
         LA    R15,8               .Stop further processing?
         ST    R15,RETCODE         .Plug the return code
         B     GETSSI@X            .Get out
ADSUBSYS EQU   *                   .First time, add subsystem
         MVC   IEFSSIA(IEFSSIL),IEFSSIM
         LA    R1,IEFSSIA
         IEFSSI REQUEST=ADD,SUBNAME=SUBSYSN,                           X
               RETCODE=RETCODE,RSNCODE=RSNCODE,MF=(E,(1))
         LTR   R15,R15             .Did we get the subsystem name?
         BNZ   SSIERROR            .No, error. Display and get out
         BAS   R14,SEEKSSI         .Get our subsystem's address
         LTR   R15,R15             .Did we get it?
         BZ    GETSSI@X            .Yes, we have the subsystem
         ABEND ØØØ1,DUMP           .Should never occur
SSIERROR BAS   R14,CODEPRNT        .No, go make RC & REASON printable
         L     R8,FROMCNID         .Where command came from
         LA    R9,FROMSYS          .Where command came from
         MVC   SUBSWTOA(SUBSWTOL),SUBSWTOM
         MVC   SUBSWTOA+42(4),PRTRC
         MVC   SUBSWTOA+57(4),PRTRSN
         LA    R1,SUBSWTOA
         WTO   MF=(E,(1)),CONSID=(8),SYSNAME=(9),CART=CART
         LA    R15,8
GETSSI@X PR
*********************************************************************
*        This routine gets the address of our subsystem entry
*********************************************************************
SEEKSSI  BAKR  R14,Ø
         L     R4,16               .Start of CVT
         USING CVT,R4              .Establish addressability
         L     R4,CVTJESCT         .Get JESCT address
         USING JESCT,R4            .Establish addressability
         L     R4,JESSSCT          .Get SSCT chain address
         USING SSCT,R4             .Set addressability
SSCTLOOP EQU   *                   .Look for our SSCVT
         LTR   R4,R4               .End of chain?
         BZ    NOSSCT              .Yes - not found
         CLC   SSCTSNAM,SUBSYSN    .Our SSCVT?
         BE    FOUNDSSI            .Got it
         L     R4,SSCTSCTA         .Point to next SSCVT entry
         B     SSCTLOOP            .Redo the loop
NOSSCT   LA    R15,8               .Our subsystem not found
         B     SEEKSSIX            .Get out
FOUNDSSI ST    R4,OURSSI@          .Keep the address
         XR    R15,R15             .Clear the return code
SEEKSSIX PR
*********************************************************************
*        This routine gets the address of the CSA table
*********************************************************************
GETCSA@  BAKR  R14,Ø               .Get address of CSA table
         L     R4,OURSSI@          .Get address of subsystem entry
         USING SSCT,R4             .Addressability to our SSI
         CLC   SSCTSUSE,=F'Ø'      .Has the CSA table been obtained?
         BE    OBTAIN              .No, go do getmain
         XR    R15,R15             .Clear return code
         MVC   CSATAB@,SSCTSUSE    .Plug the address
         B     GETCSA@X            .Get out
OBTAIN   EQU   *
         CLC   SUFFIX,=C'NO'       .Are we turning blocking off?
         BNE   CHKEMPT2            .No
         LA    R15,8               .We don't need the CSA address
         B     GETCSA@X            .Get out
CHKEMPT2 TM    EMPTY,YES           .Is the PARMLIB member anyway empty?
```

```
         BNO   PREPCSA             .No, we will have to obtain CSA
         LA    R15,8               .Yes, do not GETMAIN CSA
         ST    R15,RETCODE         .Plug the return code
         B     GETCSA@X            .Get out
PREPCSA  L     R3,BUFFSIZE         .Length of storage to obtain
         LA    R3,4(R3)            .Add 4 bytes (first word= length)
         SL    R3,=AL4(PRM_RECORD_TEXT-PRM_READ_HEADER)
         STORAGE OBTAIN,SP=228,LENGTH=(3),LOC=ANY,KEY=Ø
         LTR   R15,R15             .Did we get the storage?
         BNZ   GETCSA@X            .No, get out
         ST    R1,SSCTSUSE         .Plug the address into subsystem
         ST    R1,CSATAB@          .Plug the address
         MVC   4(13,R1),=C'Uninitialized'
         ST    R3,Ø(R1)            .Put the length in the front
GETCSA@X PR                        .Return to our caller
**********************************************************************
*        This routine removes all blanks from the PARMLIB cards and
*        copies them into the CSA table. (The size of the CSA table
*        is the same as that of the work bufer the data was read into.
*        the fact that the IEFPRMLB macro succeeded reading the data
*        into the buffer means that the data will also fit into the
*        CSA table. We cannot have a changing CSA table size as storage
*        allocated in CSA can only be freed by the task that allocated
*        it in the first place.)
**********************************************************************
COPYCRDS BAKR  R14,Ø
         CLC   SUFFIX,=C'NO'       .Are we turning blocking off?
         BNE   CHKEMPT3            .No
         XR    R1,R1               .Yes, set # entries in tab to Ø
         L     R2,CSATAB@
         ST    R1,Ø(R2)            .Make card counter Ø
         LA    R15,8               .Yes, set RC to "don't proceed"
         B     COPYCRDX            .Get out
CHKEMPT3 TM    EMPTY,YES           .Is the PARMLIB member empty?
         BNO   SETCNTR2            .No, go move the cards into the tbl
         XR    R1,R1
         L     R2,CSATAB@
SETCNTR1 ST    R1,Ø(R2)            .Make card counter Ø
         B     COPYCRDX            .Get out
SETCNTR2 LA    R2,WORKBUFF         .Where the PARMLIB cards are
         ICM   R4,15,PRM_RECORDS_READ_COUNT
         L     R5,CSATAB@          .Address of CSA table
         ST    R4,Ø(R5)            .Number of cards in first word
         LA    R5,4(R5)            .Where the cards will be moved
         LA    R2,PRM_RECORD_TEXT  .Start of the first record
CARDLOOP LR    R3,R5
         MVC   Ø(8Ø,R3),=8ØX'4Ø'   .Move spaces into the card
         LA    R6,8Ø               .Number of bytes in a card
MOVELOOP EQU   *
         CLI   Ø(R2),X'4Ø'         .Blank in the card?
         BE    BUMPUP              .Yes, don't move this byte
         MVC   Ø(1,R3),Ø(R2)       .Move the byte
         LA    R3,1(R3)             .Bump up "to" pointer
BUMPUP   LA    R2,1(R2)             .Bump up "from" pointer
         BCT   R6,MOVELOOP          .Move the entire card
         LA    R5,8Ø(R5)            .Where next card should start
         BCT   R4,CARDLOOP          .Do for each card
COPYCRDX PR
*****************************************************************
*        This routine makes the RC and reason printable
*****************************************************************
CODEPRNT BAKR  R14,Ø
         L     R1,RETCODE
         CVD   R1,DOUBLE
         UNPK  DOUBLE(4),DOUBLE+5(3)
         OI    DOUBLE+3,X'FØ'
         MVC   PRTRC,DOUBLE
         L     R2,RSNCODE
         STCM  R2,3,DOUBLE
         STCM  R2,3,DOUBLE+2
         NC    DOUBLE(2),=X'FØFØ'  .Turn off right half of bytes
         NC    DOUBLE+2(2),=X'ØFØF' Turn off left half of bytes
         TR    DOUBLE(2),LFTHALVE  .Make left half printable
         TR    DOUBLE+2(2),RGTHALVE Make right half printable
         MVC   DOUBLE+4(1),DOUBLE+1 Swap bytes 2 and 3
         MVC   DOUBLE+1(1),DOUBLE+2
         MVC   DOUBLE+2(1),DOUBLE+4 RSN code now printable
         MVC   PRTRSN,DOUBLE
         CLC   RSNCODE+2(2),=X'ØØØA' Output area too small?
         BNE   CODEPRNX             .No, other error
         L     R8,FROMCNID          .Where command came from
         LA    R9,FROMSYS           .Where command came from
         WTO   'BLOCKCMD(E): -Too many commands in PARMLIB member. reduX
               ce number of lines or assemble with larger table',      X
               CONSID=(8),SYSNAME=(9),CART=CART,ROUTCDE=11
CODEPRNX PR
****************************************************************
*        Constants follow
****************************************************************
BUFFSIZE DS    ØF
         DC    AL4(8ØØØ)
         DS    ØF
PARMDDØ1 DC    C'PARMDDØ1'          .DD-NAME used by IEFPRMLB
CALLER   DC    C'COMMAND '          .Caller name used for IEFPRMLB
IEFMACRO IEFPRMLB MF=(L,IEFPRMLB)
IEFPRMLL EQU   *-IEFMACRO
IEFSSIM  IEFSSI MF=(L,IEFSSIWA)
IEFSSIL  EQU   *-IEFSSIM
ALOCWTOM WTO   'BLOCKCMD(E): -Error during allocation/ reading of parmlX
               ib member,RC=XXXX(DEC), RSN=xxxx',MF=L
ALOCWTOL EQU   *-ALOCWTOM           .Length of the message
FREEWTOM WTO   'BLOCKCMD(E): -Error during free of PARMLIB member, RC=xX
               xxx(DEC), RSN=xxxx',MF=L
FREEWTOL EQU   *-FREEWTOM           .Length of the message
```

```
SUBSWTOM WTO   'BLOCKCMD(E): -Error during IEFSSI, RC=xxxx(dec), RSN=xxX
               xx',MF=L
SUBSWTOL EQU   *-SUBSWTOM           .Length of the message
LFTHALVE DS    ØCL24Ø
         DC    X'FØ',15X'ØØ',X'F1',15X'ØØ',X'F2',15X'ØØ',X'F3'
         DC    15X'ØØ',X'F4',15X'ØØ',X'F5',15X'ØØ',X'F6',15X'ØØ',X'F7'
         DC    15X'ØØ',X'C3',15X'ØØ',X'C4',15X'ØØ',X'C5',15X'ØØ',X'C6'
VALCHARS DS    ØCL256
         DC    8ØX'Ø1',X'ØØ',1ØX'Ø1',X'ØØ',16X'Ø1',2X'ØØ'
         DC    13X'Ø1',2X'ØØ'
         DC    68X'Ø1',9X'ØØ',7X'Ø1',9X'ØØ',8X'Ø1'
         DC    8X'ØØ',6X'Ø1',1ØX'ØØ',6X'Ø1'
RGTHALVE DC    X'FØF1F2F3F4F5F6F7F8F9C1C2C3C4C5C6'
SUBSYSN  DC    C'NCMD'              .Name of our subsystem entry
         LTORG
*********************************************************************
*        DSECTS follow
*********************************************************************
STORAREA DSECT
SAVEAREA DS    18F                  .General savearea
PARMSTRT DS    F                    .Start address of passed parms
FROMCNID DS    CL4                  .Console id command came from
FROMSYS  DS    CL8                  .Name of system command came from
RETCODE  DS    F                    .Return code
RSNCODE  DS    F                    .Reason code
PRTRC    DS    F                    .Return code (printable)
PRTRSN   DS    F                    .Reason code (printable)
OURSSI@  DS    F                    .Address of our subsystem
CSATAB@  DS    F                    .Address of the CSA table
MEMNAME  DS    ØCL8
MEMBER   DS    CL5                  .This contains "NOCMD"
SUFFIX   DS    CL2                  .Suffix of PARMLIB member to obtain
LASTBYTE DS    C                    .This will contain a blank
EMPTY    DS    C                    .Flag to indicate PARMLIB empty
DOUBLE   DS    D                    .Double word work area
CART     DS    CL8
ALOCWTOA DS    CL(ALOCWTOL)         .Work area for WTO message
FREEWTOA DS    CL(FREEWTOL)         .Work area for WTO message
SUBSWTOA DS    CL(SUBSWTOL)         .Work area for WTO message
IEFPRMLA DS    CL(IEFPRMLL)         .Work area for IEFPRMLB macro
IEFSSIA  DS    CL(IEFSSIL)          .Work area for IEFSSI macro
WORKBUFF EQU   *                    .Parmlib buffer area
STORSIZE EQU   *-STORAREA           .Length of area to allocate
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
NO       EQU   X'ØØ'
YES      EQU   X'8Ø'
         IEZVX1Ø1                  .DSECT for command exit fields
         IEFZB4DØ
         IEFZB4D2
         IEFZPMAP  PRM_READ_BUFFER=YES
         CVT   DSECT=YES
         IEFJESCT
         IEFJSCVT
         IEFJSRC
         IEFJSQRY
         END
```

Gerty Brits
Minalore Consulting (India)

OS/390 Unix Assembler callable services

With the introduction of OpenEdition, or OS/390 Unix System Services as it is now called, IBM has provided a rich and varied set of program callable services that are available to both Assembler language and TSO/E REXX programmers. The main objective of this article is to introduce the reader to the OS/390 Unix callable services that are available to Assembler language programs and to also make reference to REXX callable services where applicable.

The first section of this article provides an overview of OS/390 Unix process management, which I have found to be an important learning stage before attempting to write Unix-style Assembler programs or REXX EXECs.

PROCESS MANAGEMENT

In OS/390 Unix, a child process is created when the parent process issues a fork service call. The creating process is called a parent process and the newly created process is called a child process. A parent process can have many child processes, the number of which is controlled by the MAXPROCUSER statement in the BPXPRMxx PARMLIB member. Each process is given a unique identifier called the Process ID (PID). Each process also knows the ID of its parent through the Parent Process ID (PPID). All processes are related to each other through the PIDs and the PPIDs. The originator of all processes is called the INIT process (PID = 1). In addition to having a PID, each process belongs to a process group. A process group is a collection of one or more processes. Each process group has a unique group ID. Process identifiers are classified as follows:

- PID – a process ID. A unique identifier assigned to a process while it runs.
- PGID – each process in a process group shares a process group ID (PGID), which is the same as the PID of the first process in the process group.
- PPID – a process that creates a new process is called a parent process.

Process management can be broken down into the following areas:

- Processes
- Dubbing
- Threads
- Interprocess communication
- Signals.

A PROCESS

A process exists in an MVS address space and is identified by a TCB and related control blocks. In addition to the TCB, the OS/390 Unix kernel address space maintains a number of control blocks that represent a process. The following BPXPRMxx PARMLIB member statements control OS/390 Unix processes:

- MAXPROCSYS(nnnnn) – specifies the maximum number of OS/390 Unix processes that the system allows.
- MAXPROCUSER(nnnnn) – specifies the maximum number of processes that a single OS/390 Unix user-id can have currently active, regardless of how the processes were created.

To control processes, the following Assembler callable services are available.

BPX1FRK (fork service)

BPX1FRK creates a new process. The fork service replicates the current process into a child process, which then runs in a new address space. The new address space contains a single task (thread) and a single RB (Request Block) structure. The fork service is supported from programs in PSW key 8 only. An additional requirement is that the storage protection key value in the TCBPKF field of the TCB must be 8. All key 8 virtual storage is copied from the parent to the child address space. The child process has a unique PID that does not match any active process group ID. The child has its own copy of the parent's open directory stream. The fork service can be requested from either an MVS or kernel address space. Control is given to the child process at the instruction following the fork service call and not at the program's main entry point. In most implementations, the parent process will continue and the child process will pass control using the EXEC service to a child-specific program. The BPXPRMxx PARMLIB member statement FORKCOPY(COW|COPY) specifies how user storage is to be copied from the parent process to the child process during a fork system call.

BPX1SPN (spawn service)

BPX1SPN spawns a process. The spawn service starts a new process, but the child process is started with another program in the Hierarchical File System (HFS). After the spawn service returns to the parent process, the two processes continue as independent processes. The main benefit of the spawn callable service is that it can create a new process in a separate address space or in the same address space, depending on the setting of the environment variable '_BPX_SHARES=YES|NO'. If an application is multi-threaded, you must use spawn instead of the fork callable service. There are some exceptions where, despite setting '_BPX_SHAREAS=YES', a non-local spawn (child process starts in another address space) is done. A non-local spawn is done in any of the following cases:

- The program that is spawned has the sticky bit on
- The program that is spawned is a SETUID or SETGID program
- The address space has exhausted its private storage.

A new setting of '_BPX_SHAREAS=MUST' was added in OS/390 Version 2 Release 6. It allows the application to force a local spawn or no spawn at all. The new process, which is called the child process, inherits the following attributes from the parent process (process that calls spawn):

- Session membership.
- Real user-id.
- Real group-id.
- Supplementary group-ids.
- Priority.
- Region size.
- Time limit.
- Accounting data.
- Working directory.
- Root directory.
- File creation mask.
- Signal mask.
- Security information, unless the '_BPX_USERID' environment variable specifies otherwise.
- TASKLIB STEPLIB or JOBLIB DD dataset allocations, unless the STEPLIB environment variable specifies otherwise. This causes the child's address space to have the same MVS program search order as the calling process.

- Accounting information.
- JOBNAME of the parent is propagated to the child and appended with a numeric value in the range 1-9, if the JOBNAME is 7 characters or less. If the JOBNAME is 8 characters, the JOBNAME is propagated as is.

The executable program to be run receives control with the following attributes:

- Problem program state
- PSW key 8
- AMODE=31
- Primary ASC mode.

BPX1ATX (attach_exec) service

BPX1ATX attach an OS/390 Unix program. The 'attach_exec' service attaches a task to run an OS/390 Unix executable program in a newly created child process of the caller. The new process is created in the same address space as the caller, and is a subtask of the caller's task. The child process has a unique PID that does not match any active process group ID. The child process has a parent process ID of the process that called attach_exec. The child process is terminated when its parent terminates. The executable file receives control with the following attributes:

- Problem program state
- TCB key of the caller
- AMODE=31
- Primary ASC mode.

The equivalent function is provided by the BPX1SPN SPAWN Service with '_BPX_SHAREAS=YES'. The OMVS command uses the attach_exec system call to run the shell in the TSO/E address space.

BPX1ATM (attach_execmvs) service

BPX1ATM ATTACHes an MVS program. The attach_exec MVS service creates a new child process in the same address space and passes control to a new program in the normal MVS search order (job pack queue, STEPLIB, LPALIB, LINKLIB). The new process that is created is run as a subtask in the address space.

BPX1EXC (EXEC) service

BPX1EXC run a program. The EXEC service does not start a new process, but replaces the program in the current process with another program as indicated on the EXEC call. A successful EXEC call will never return control to the calling program, but control is passed to the main entry point of the new program that is specified on the EXEC call. The executable program receives control with the following attributes:

- Problem program state
- PSW key 8
- AMODE=31
- Primary ASC mode.

The new process inherits the following attributes from the calling process:

- PID
- PPID
- The time left until an alarm signal is generated
- File mode creation mask
- Process signal mask
- Pending signals
- Time accounting information.

BPX1EXM (execmvs) service

BPX1EXM run an MVS program. The execmvs service runs an MVS executable program that is in the LPA or LNKLST concatenation. If it is invoked from an address space that contains multiple processes, the program can come from a STEPLIB. The call can invoke both unauthorized and authorized programs:

- Unauthorized programs receive control in problem program state, with PSW key 8.
- Authorized programs receive control in problem program state, with PSW key 8 and APF authorization.

Additional Assembler process callable services

The following additional Assembler callable services are available:

•	chpriority (BPX1CHP)	–	change the scheduling priority of a process
•	getpid (BPX1GPI)	–	get the process ID
•	getppid (BPX1GPP)	–	get the parent process ID
•	getpgid (BPX1GEP)	–	get the process group ID
•	getpgrp (BPX1GPG)	–	get the process group ID
•	getpriority (BPX1GPY)	–	get the scheduling priority of a process
•	nice (BPX1NIC)	–	change the nice value of a process
•	setpgid (BPX1SPG)	–	set a process group ID for job control
•	setpriority (BPX1SPY)	–	set the scheduling priority of a process
•	_pid_affinity (BPX1PAF)	–	add or delete an entry in a process’s affinity list.

REXX PROCESS MANAGEMENT

There is no FORK or EXEC SYSCALL commands available in REXX. Instead IBM recommend using SPAWN. The following REXX process management calls are available:

Spawn

Spawn invokes the spawn callable service to create a new process, called the child process. The program to be run is executed from the HFS. The child process has a unique PID that does not match any active process group ID. The child parent ID is set to the PID of the process that called spawn. To control whether the spawned child process runs in a separate address space or in the same address space, you can specify the '_BPX_SHAREAS' environment variable. If '_BPX_SHAREAS=NO' is specified, the child process to be created will run in a separate address space from the parent process. If '_BPX_SHAREAS=YES' is specified, the child process to be created is to run in the same address space as the parent. The following attributes are inherited by the child process from the parent process:

- Session membership
- Real user-id
- Real group ID
- Supplementary group IDs
- Priority
- Working directory
- Root directory
- File creation mask
- The process group ID of the parent is inherited by the child
- Signals set ignored in the parent are set to be ignored in the child
- The signal mask is inherited from the parent.

Example

The following example will spawn a new process. The new process is spawned to run /bin/ls. File descriptors greater than or equal to three are not available to the new process. fd.0 to fd.2 are used in remapping the file descriptors from the parent. The current environment is passed to the new process.

```
address syscall
/* Initialize the file descriptor map. The file descriptor map is set so */
/* that the file descriptor for the new file being created is remapped   */
/* to file descriptor 1 for the new process. File descriptors 0 and 2    */
will not be opened in the new process (-1).                              */

fd.Ø=-1
fd.2=-1
/* Create a new HFS file. The file descriptor is returned in RETVAL      */
‘creat /tmp/remdirl 755’
fd.1=retval
/* Initialize the parameter stem. The first parameter is set to the      */
pathname for the file being spawned. Additional parameters are           */
/* are set in the format the program expects. The parameters specif      */
/*    Display extended attributes for regular files                      */
/*    Display permissions, links, owner, group, size, time, name         */
/*    Enables the audit bits to be displayed                             */
/*    The current environment is propagated (__environment)              */
parm.1=‘/bin/ls’
parm.2=‘-lWE’
parm.3=‘/u/rems’
parm.0=3
/* spawn a new process */
‘spawn /bin/ls 3 fd. parm. __environment’      /* PID(process ID) is returned
in RETAIL */
pid=retval
```

spawnp

Spawnp invokes the spawn callable service and creates a new process, called a child process. Spawnp functions identically to the spawn function except that it uses the PATH environment variable to resolve relative filenames.

forkexecm

Forkexecm invokes the fork and execmvs callable services to fork and EXEC a program to be executed from the MVS LINKLIB, LPALIB, or STEPLIB library. The call can invoke both authorized and unauthorized MVS programs. Authorized programs receive control in problem program state with PSW key 8 and APF authorization. Unauthorized programs receive control in problem state, with PSW key 8.

Example

The following example invokes the MVS program unixprog and passes a parameter to the program. On input, the MVS program receives a single entry parameter list pointed to by Register 1. The high order bit of the sole parameter entry is set to 1.

```
"forkexecm unixprog 'This is the parm info'
```

Dubbing

The first attempt to use a OS/390 Unix services dubs the MVS address space as an OS/390 Unix process. From a Unix perspective, this means OS/390 Unix assigns a PID (Process ID) to the process. Address spaces created by fork are automatically dubbed when they are created. Dubbing also adds the UID/GID assignment to an address space as follows:

Real UID

At process creation, the real UID identifies the user who has created the address space.

Effective UID

Each process has an effective UID. The effective UID is used to determine owner access privileges of a process. This is normally the same as the real UID but can be changed when a program is executed that has a special flag. A program with this special flag set is said to be a set-user-id program. This changes the effective UID of the process to the UID of the owner of the program, to allow additional permissions to the user while the set-user-id program is executed.

Real GID

At process creation , the real GID identifies the current connect group of the user for which the process was created.

Effective GID

Each process has an effective GID. The effective GID is used to determine group access privileges of a process. This is normally the same as the real GID but can be changed when a program is executed which has a special flag. A program with this special flag is said to be a set-group-id program. This changes the effective GID of the process to the GID of the owner of the program, to allow additional permissions to the user while the set-group-id program is executed. Undub is the inverse of dub. Normally a task (dubbed a thread) is undubbed when it ends. An address space (dubbed a process) is undubbed when the last thread ends.

THE REXX SYSCALL ENVIRONMENT

IBM has provided two additional host command environments in REXX as follows:

- SYSCALL
- SH

To run a REXX program with SYSCALL commands from TSO/E or MVS batch, the syscalls (‘ON’) function at the beginning of the REXX program ensures that the address space is dubbed an OS/390 Unix process. For a REXX program that utilizes SYSCALL commands from the shell, SH is the initial host environment. The SYSCALL environment is automatically initialized as well, so there is no need to begin the REXX program with a syscalls(‘ON’) call. The SYSCALL environment sets up the REXX pre-defined variables and blocks all signals. The syscalls(‘ON’) function sets the following return code values:

0 Successful completion.

4 The signal process mask was not set.

7 The process was dubbed, but the SYSCALL environment was not established.

8 The process could not be dubbed.

The following example shows how you can use the syscalls (‘ON’) function at the beginning of a REXX program to establish the SYSCALL environment and get the address space dubbed as an OS/390 Unix process:

```
If syscalls('ON') >3 Then Do
   say 'Unable to establish the SYSCALL environment'
   Return
End
```

Or as an alternative:

```
/* REXX*/
address syscall ‘pipe p.’    /* Make a pipe         */
‘ls>/dev/fd’ || p.2         /* run ls command and redirect output to the
                              write end of the pipe */
address syscall ‘close’ p.2    .* close output side   */
```

To end the REXX SYSCALL environment the syscalls (‘OFF’) function ends the current task and OS/390 Unix process. The following rules apply:

- If the REXX program was run from TSO/E or batch, the task is undubbed, but the REXX program continues running.
- If the REXX program was run from the shell or a program, the REXX program is ended.

Threads

In POSIX, a thread is an entity that allows multiple simultaneous execution paths within a process. The OS/390 Unix design implementation for creating a thread is to attach a TCB within a process (address space). Threads allow multiple tasks to run in a single process within an address space. It allows for concurrent and asynchronous processing without the additional overhead associated with creating a new address space. Each thread of a process can run on an individual processor in a multi-processor environment. When using Assembler as opposed to the C language, it easier to do an ATTACH and let the tasks be dubbed as threads. Threads are created as follows:

- The pthread_create service
- The fork or EXEC service
- Most OS/390 Unix service requests from an undubbed MVS task.

The first routine that is given control in the new task when a thread is created with the pthread_create service is the pthread_create pthread-creating task initialization routine. Each thread that is created with pthread_create runs as a MVS subtask of the initial pthread_creating task (IPT). The IPT is the task that issued the first pthread_create call within the address space. Threads created by pthread_create are represented by a eight-character thread ID. There are three thread types.

Heavy-weight threads

A heavy-weight thread has been defined as a task that is attached when needed. A heavy-weight thread is created by issuing the pthread_create system call and specifying PTATWEIGHT= PTATHEAVY in the BPXYPTAT mapping macro. When a heavy-weight thread terminates, the task (TCB) that supports it is terminated and all EOT (end_of_task) resource managers are called to clean up after it.

Medium-weight threads

Medium-weight threads reuse MVS tasks. A medium-weight thread is created by issuing the pthread_create system call and specifying PTATWEIGHT= PTATMEDIUM in the BPXYPTAT mapping macro.

When a medium-weight thread is created, it is dispatched using an MVS task that is maintained in a pool. When a medium-thread terminates using pthread_exit, the MVS task is recycled in the pool without going through the MVS EOT (end_of_task) resource managers. OS/390 Unix reuses the task. An example of an OS/390 Unix application that uses medium-weight threading is the OS/390 Internet Connection Server.

Light-weight threads

Light-weight threads have not yet been implemented in LE/390 and OS/390 Unix. Thread manipulation is available using the following OS/390 Unix Assembler callable services:

- pthread_create (BPX1PTC) – create a thread
- pthread_cancel (BPX1PTB) – cancel a thread
- pthread_detach BPX1PTD) – detach a thread
- pthread_exit_and_get (BPX1PTX) – exit and GET a new thread
- pthread_join (BPX1PTJ) – wait on a thread
- pthread_kill (BPX1PTK) – send a signal to a thread
- pthread_quiesce (BPX1PTQ) – quiesce threads in a process
- pthread_self (BPX1PTS) – query the thread ID
- pthread_security_np (BPX1TLS) – create/delete thread-level security environment for callers thread. An installation has the

following two ways of allowing an application to use this service:

  - Define the BPX.SERVER FACILITY class profile. For an application to access this service, it must be given read access to this profile.
  - Assign the user-id a UID of 0 so that it operates as a superuser.

- pthread_setintr (BPX1PSI) – examine and change the interrupt state
- pthread_setintrtype (BPX1PST) – examine and change the interrupt type
- pthread_tag_np (BPX1PTT) – set, query, or both set and query the callers thread tag data.
- pthread_testintr (BPX1PTI) – cause a cancellation point to occur.

Please refer to the *Unix System Services Programming: Assembler Callable Services Reference* for a detailed description of the thread callable services.

OS/390 Unix threads are controlled by the following BPXPRMxx PARMLIB member statements:

- MAXTHREADTASKS(nnnnn) – specifies the maximum number of MVS tasks that a single process can have concurrently active for pthread_created threads.
- MAXTHREADS(nnnnn) – specifies the maximum number of pthread_created threads, including running, queued, and exited but undetached, that a single process can have concurrently active.

Signals

In OS/390 Unix applications, the basis for error handling is the generation, delivery, and handling of signals. Each process has a signal mask that defines the set of signals currently blocked from delivery and that is inherited by a child from its parent. Applications can be coded to generate and send signals, and to handle and respond to signals delivered to it. During the time between the generation of a signal and the delivery of a signal (when the actual signal is performed), the signal is said to be pending. It is valid for the process to block the signal. If a signal that is blocked is generated for a process and the action for that signal is either the default action or to catch the signal, the signal remains pending for the process until the process either unblocks the signal or changes the action to ignore the signal.

The signal mask, which is passed as a parameter using the Assembler callable services, is structured as a 64-bit mask (8 bytes) of signals that are to be blocked during execution of the signal-catching function. The leftmost bit represents signal number 1, and the rightmost bit represents signal number 64. Bits that are set to 1 represent signals that are blocked. In the REXX callable services, the signal mask is defined as a string of 64 characters with values 0 or 1, representing the 64 bits in a signal mask.

Example

Re-sets the action for signals 5-64 to their default state:

```
        X'ØFFFFFFFFFFFFFFF'
```

Ignore signals 1-4

```
        X'FØØØØØØØØØØØØØØØ'
```

The following OS/390 Unix SIGNAL callable services are available for the Assembler language programmer.

BPX1SIA (sigaction service)

BPX1SIA examines or changes a signal action. The sigaction service examines, changes or both examines and changes the action that is associated with a specific signal for all threads in the process.

BPX1SA2 (_sigactionset service)

BPX1SA2 examines or changes a set of signal actions. The _sigactionset service examines, changes or both examines and changes the actions that are associated with a set of signals.

BPX1SIP (sigpending service)

BPX1SIP examines pending signals. The sigpending service returns the union of signals that are pending on the thread and the set of signals that are pending on the process.

BPX1SIP (sigprocmask service)

BPX1SIP examines or changes a process’s signal mask. The sigprocmask service examines, changes or both examines and changes the actions that are associated with a set of signals.

BPX1SSU (sigsuspend service)

BPX1SSU changes the signal mask and suspends the thread until a signal is delivered. The sigsuspend service replaces a thread’s current signal mask with a new signal mask. It then suspends the caller’s thread until delivery of a signal whose action is either to process a signal-catching service or to end the thread.

BPX1SWT (sigwait service)

BPX1SWT waits for a signal. The sigwait service waits for an asynchronous signal. If a signal that is specified in the signal set is sent to the invoker of sigwait, the value of that signal is returned to the invoker and the sigwait service ends. The following signals are supported:

SIGHUP	(1)	Hang-up detected on controlling terminal
SIGINT	(2)	Interactive attention
SIGABRT	(3)	Abnormal termination
SIGILL	(4)	Illegal or invalid hardware instruction
SIGPOLL	(5)	Pollable event
SIGURG	(6)	High bandwidth data is available at a socket
SIGSTOP	(7)	Stop executing
SIGFPE	(8)	Erroneous arithmetic operation (hardware and software)
SIGKILL	(9)	An unconditional terminating signal
SIGBUS	(10)	Bus error
SIGSEGV	(11)	Invalid access to memory (hardware and software)
SIGSYS	(12)	Bad system call

- SIGPIPE (13) Write on a pipe with no readers
- SIGALRM (14) Asynchronous time-out signal generated as a result of an alarm()
- SIGTERM (15) Termination
- SIGUSR1 (16) Reserved as an application-defined signal 1 (Software only)
- SIGUSR2 (17) Reserved as an application-defined signal 2 (Software only)
- SIGABND (18) Abend
- SIGCONT (19) Continue if stopped
- SIGCHLD (20) Child process terminated or stopped
- SIGTTIN (21) A background process is attempting a read
- SIGTTOU (22) A background process is attempting a write
- SIGIO (23) Completion of input or output
- SIGQUIT (24) Interactive termination
- SIGTSTP (25) Interactive stop
- SIGTRAP (26) Trap used by ptrace call
- SIGIOERR (27) I/O error. Serious software error such as a system read or write.
- SIGWINCH (28) Change size of window
- SIGXCPU (29) CPU time limit exceeded
- SIGXFSZ (30) File size limit exceeded
- SIGVTALARM (31) Virtual timer expired
- SIGPROF (32) Profiling timer expired
- SIGDCE (38) Exclusive use by DCE.

For a full description of signals that are available please refer to the mapping macro *BPXYSIGH – Signal Constants* in the *Unix System Services Programming: Assembler Callable Services Reference*.

The signals SIGSTOP and SIGKILL cannot be blocked or ignored, they are delivered to the program no matter what the signal mask specifies. Please note that the use of the system linkage stack with PC and BAKR instructions prevents signals from being delivered.

REXX SIGNAL SERVICES

The following REXX signal services are available:

- Sigaction – this invokes the sigaction callable service to examine or change, or both, the action associated with a specific signal for all the threads in the process.
- Sigpending – this invokes the sigpending callable service to return the union of the set of signals that are pending on the thread and the set of signals pending on the process. Pending signals at the process level are moved to the thread that called the sigpending callable service.
- Sigprocmask – this invokes the sigprocmask callable service to examine or change the calling thread's signalmask.
- Sigsuspend – this invokes the sigsuspend callable service to replace a thread's current signal mask with a new signal; then it suspends the caller's thread until delivery of a signal whose action is either to process a signal-catching service or to end the thread.
- Sleep – this invokes the sleep callable service to suspend running of the calling thread (process) until the number of seconds has elapsed (*sleep* – number), or until a signal is delivered to the calling thread to invoke a signal-catching function or to end the thread.
- Alarm – this invokes the alarm callable service to generate a SIGALRM signal after the number of seconds specified has elapsed (*alarm* – seconds)
- Kill – this invokes the kill callable service to send a signal to a process or process group (*kill* – pid – signal).
- Pause – this invokes the pause callable service to suspend execution of the calling thread until delivery of a signal that either executes a signal-catching function or ends the thread.

Establishing and deleting the signal interface routine in REXX

The syscalls (‘SIGON’) function establishes the Signal Interface Routine (SIR). For a REXX program run from the shell or a program, the SIR is established by default. After the SIR has been established, the sigaction syscall command can be issued to catch the signals that are to be processed and the sigprocmask syscall command can be used to unblock the signals. The syscalls (‘SIGOFF’) function deletes the signal interface routine.

The following example enables the sigalarm signal. Sigaction is used to set the action for the sigalrm to be caught. Sigprocmask is used to unblock sigalrm. The alarm is set by using the alarm service and the process waits for the completion of the child or the alarm:

```
/*REXX */
address syscall
/* Insert instructions to spawn a new process */
.......
.......
pid=retval
/* Child process ID from spawn                                     */
call syscalls ('SIGON')
/* Establish the signal interface routine                          */
'sigaction' sigalrm sig_cat 0 'old_ handler old_flag'
/* catch a SIGALRM signal                                          */
'sigprocmask'  sig_unblock sigaddset(sigsetempty(),sigalrm) 'mask'
 /* use sigaddset and sigsetempty to create a signal               */

/* mask with the sigalrm bit                                       */
'alarm 20'
/* set alarm to expire in 20 seconds                               */
'waitpid (pid) st. 0'
/* Wait for process termination or alarm                           */
alarm_ind=retval
/* Status                                                          */
/* Check if alarm went off                                         */
.......
.......
call syscalls ('SIGOFF')         /* Turn off signals               */
/* Determine process status using stem variable st. returned from
waitpid */
.........
.........
Exit
/* All done                                                        */
sigsetempty: return copies(0,64)
sigaddset: return overlay(1,arg(1),arg(2))
```

REXX statements for defining signal sets

REXX statements for defining signal sets are shown below. The C function is shown first followed by the equivalent REXX statement with its parameters and returns.

sigsetempty()	sigsetempty: return copies(0,64) Parameters: none Returns: signal set
sigsfillset()	sigfillset: return copies(1,64) Parameters: none Returns: signal set
sigsaddset()	sigaddset: return overlay(1,arg(1),arg(2)) Parameters: signal set, signal number Returns: signal set
sigsdelset()	sigdelset: return overlay(0,arg(1),arg (2)) Parameters: signal set, signal number Returns: signal set
sigismember()	sigismember: return substr(arg(1),arg(2),1) Parameters: signal set, signal number Returns: 0 (not member) or 1 (is member).

Refer to the REXX example in the section entitled *Establishing and Deleting the Signal Interface Routine in REXX*, for the sigsetempty and sigaddset REXX statements.

DETERMINING THE OS/390 CALLABLE SERVICE LEVEL

The Aassembler language programmer can determine the OS/390 Unix release-level by interrogating the CVT feature flags. At the time of writing the following values are defined:

- CVTH6603 EQU X'04' HBB6603 (OS/390 Release 3) functions are present
- CVTH6605 EQU X'40' HBB6605 (OS/390 Release 5) functions are present
- CVTH6606 EQU X'20' HBB6606 (OS/390 Release 6) functions are present.

Assembler callable services syntax

To code a callable service, a CALL macro followed by the name of the callable service and parameter list is required. The required syntax is shown below:

```
      CALL  Service_Name,(PARM_1,
                          PARM_2,
                          .
                          .
                          Return_value,
                          Return_code,
                          Reason_code)
```

CALL — CALL is the Assembler macro that passes control to the specified program and passes a parameter list.

Service_Name — Call service module name in the form BPX1xxx, where: 'xxx=' is a three-character symbol that is unique to the service. An example would be BPX1CHM=chmod service.

PARM parameters PARM_1, PARM_2, etc, are placeholders for variables that may be part of a service syntax.

Return_value — Indicate the success failure of the callable service.

If the callable service fails, a -1 is returned. For most successful calls to OS/390 Unix Services, the return value is set to 0. The BPX1GGI and BPX1GGN callable services return zeroes instead of -1 when the service fails. The fork (BPX1FRK) callable service returns a positive return value to indicate successful invocation.

Return_code — The return_code parameter is referred to as the errno in the POSIX and X/OPEN C interface. The Return Code is returned only if the service fails. All the Return codes and descriptions can be found in *OS/390 Unix System Services Messages and codes*.

Reason_code — The Reason_code parameter usually accompanies the Return_code value when the callable service fails. It further defines the return_code. All the Reason codes and their descriptions can be found in the *OS/390 Unix System Services Messages and Codes*.

Linkage conventions

The following linkage conventions are used when invoking the Callable Services:

- R1 Parameter list address. The last word in the list must have a 1 in the high order bit (Sign bit)
- R13 Savearea address
- R14 Return address
- R15 Entry point address of the service stub that is being called.

Additional notes

1 R2-R13 are restored on return from a callable service. R0, R1, R14, R15 are not restored.

2 The caller must be running with 31-bit addressing mode (AMODE=31).

Mapping macros

Many of the Assembler callable services provide mapping macros to map the parameter options. Many can be expanded with or without a DSECT statement. Please refer to the *Unix System Services Programming: Assembler Callable Services Reference* for the complete list of mapping macros that are available.

OS/390 UNIX ASSEMBLER CALLABLE SERVICES

The following list of OS/390 Unix Assembler Callable Services is a subset of what is currently available. Please refer to the *Unix System Services Programming: Assembler Callable Services Reference* for the complete list.

- access (BPX1ACC) – determine if a file can be accessed.
- asyncio (BPX1AIO) – asynchronous I/O for sockets.
- auth_check_resource_np (BPX1ACK) – determine a user’s access to protected MVS resource. The authorization required to invoke this service is one of the following:
  - Read access to the BPX.SERVER FACILITY class profile

  - A UID of 0 when the BPX.SERVER Facility class profile is not defined.

- chattr (BPX1CHR) – change the attributes of a file or directory.
- chaudit (BPX1CHA) – change audit flags for a file by path.
- chdir (BPX1CHD) – change the working directory.
- chmod (BPX1CHM) – change the mode of a file or directory.
- chown (BPX1CHO) – change the owner or group of a file directory.
- chroot (BPX1CRT) – change the root directory.
- close (BPX1CLO) – close a file.
- fchattr (BPX1FCR) – change the attributes of a file or directory by descriptor.
- fchaudit (BPX1FCA) – change audit flags for a file by descriptor. fchdir (BPX1FCD) – change the working directory.
- fchmod (BPX1FCM) – change the mode of a file or directory by descriptor.
- fchown (BPX1FCO) – change the owner and group of a file or directory by descriptor.
- fstat (BPX1FST) – get status information about a file by descriptor.
- getcwd (BPX1GCW) – get the pathname of the working directory.
- getegid (BPX1GEG) – get the effective group ID.
- geteuid (BPX1GEU) – get the effective user-id.
- getgid (BPX1GID) – get the real group ID.
- getgroups (BPX1GGR) – get a list of supplementary group IDs.
- getgroupsbyname (BPX1GUG) – get a list of supplementary group IDs by user name.
- getpwnam (BPX1GPN) – access the user database by user name.
- getpwuid (BPX1GPU) – access the user database by user-id.

- getuid (BPX1GUI) – get the real user-id.
- getpwd (BPX1GWD) – get the pathname of the working directory.
- lchown (BPX1LCO) – change the owner or group of a file, directory, or symbolic link.
- link (BPX1LNK) – create a link to a file.
- loadhfs (BPX1LOD) – load a program into storage by HFS pathname.
- lstat (BPX1LST) – get status information about a file or symbolic link by pathname.
- mkdir (BPX1MKD) – make a directory.
- mount (BPX1MNT) – make a file system available.
- mknod (BPX1MKN) – make a directory, a FIFO, a character special, or a regular file.
- open (BPX1OPN) – open a file.
- opendir (BPX1OPD) – open a directory.
- openstat (BPX2OPN) – opens a file, creates a file descriptor for it, and obtains its status.
- pipe (BPX1PIP) – create an unnamed pipe.
- quiesce (BPX1QSE) – quiesce a file system.
- read (BPX1RED) – read from a file.
- read_extlink (BPX1RDX) – read an external symbolic link.
- readdir (BPX1RDD) – read an entry from a directory.
- readdir2 (BPX1RD2) – read multiple entries from a directory.
- realpath (BPX1RPH) – resolve a pathname.
- rename (BPX1REN) – rename a file or directory.
- resource (BPX1RMG) – obtain system-wide resource management data from the kernel address space.
- set _dub_default (BPX1SDD) – Get the dub default service.

- setegid (BPX1SEG) – set the effective GID.
- seteuid (BPX1SEU) – set the effective user-id.
- setgid (BPX1SGI) – set the GID.
- setgroups (BPX1SGR) – set the supplementary group IDs list.
- setitimer (BPX1STR) – set the value of the interval timer.
- setpgid (BPX1SPG) – set a process GID for job control.
- setregid (BPX1SRG) – set the real and/or effective GIDs.
- setuid (BPX1SUI) – set user-ids.
- stat (BPX1STA) – get status information about a file by pathname.
- symlink (BPX1SYM) – create a symbolic link to a pathname.
- truncate (BPX1TRU) – change the size of a file.
- ttyname (BPX2TYN) – (X/Open version) get the name of a terminal.
- unmask (BPX1UMK) – set the file mode creation mask.
- uname (BPX1UNA) – obtain the name of the current operating system.
- unlink (BPX1UNL) – remove a directory entry.
- unquiesce (BPX1UQS) – unquiesce a file system.
- utime (BPX1UTI) – set file access and modification times
- _wlm (BPX1WLM) – WLM interface service.
- write (BPX1WRT) – write to a file.

There are also a number of callable services that deal with:

- Socket processing
- Semaphores
- Memory mapping
- Message queue processing.

Callable Service Request Table (CSRTABLE)

When link-editing an Assembler module that contains OS/390 Unix Callable Services, a service stub must be included as part of the load module. The library containing the service stubs is specified in the SYSLIB concatenation in the linkage editor step. An alternative to this method is to use the system control offsets to callable services. The Callable Service Request Table (CSRTABLE), whose address is contained in the CVT provides the addresses of all the callable services. To locate the required callable service, an offset into this table is required. The full list of offsets is available in the *Unix System Services Programming: Assembler Callable Services Reference*.

When using the offsets, the registers must be set as follows:

- Register 1 To contain the address of the parameter list. Bit 0 of the last address in the list must be set on.
- Register 14 To contain the return address in the invoking module.
- Register 15 To contain the address of the callable service code.

The following example locates and executes the chown service:

```
     L    R15,16          Address of the CVT
USING     CVT,R15         Obtain addressability to The CVT
     L    R15,CVTCSRT     Pointer to the Callable Service Request Table
     L    R15,24(,R15)    CSR slot
     L    R15,64(,R15)    Chown slot entry = 64
BALR      R14,R15         Execute the callable service
```

OS/390 UNIX ASSEMBLER CALLABLE SERVICES EXAMPLES

```
***********************************************************************
* THIS EXAMPLE OPENS, READS AND CLOSES THE ROOT DIRECTORY             *
* SERVICES : OPENDIR READDIR CLOSEDIR                                 *
***********************************************************************

OPEN_READ_CLOSE_DIR  EQU *
         MVC   CALLAREA(CALLLEN),CALLL MAKE RENT
         MVC   DIR_LEN,=AL4(L'ROOT)    ROOT LENGTH
         MVC   DIR_NAME(L'ROOT),ROOT   ROOT
OPEN_DIR EQU *
         CALL  BPX1OPD,                OPENDIR                         X
               (DIR_LEN,               DIRECTORY NAME  LENGTH          X
               DIR_NAME,               DIRECTORY NAME                  X
               RETVAL,                 RETURN VALUE:-1 OR FD           X
               RETCODE,                RETURN CODE                     X
               RSNCODE),               REASON CODE                     X
               VL,                                                     X
               MF=(E,CALLAREA)
         ICM   R15,B'1111',RETVAL      TEST  RETVAL
         BL    OPEN_DIR_ABEND          BRANCH IF NEGATIVE (-1= FAILURE)
         STCM  R15,B'1111',DIR_DESCP   STORE THE DIRECTORY
READ_DIR EQU *
         LA    R15,DIR_READ_BUFFER     DIR READ BUFFER ADDRESS
         STCM  R15,B'1111',DIRREAD_BUFFER@  STORE AWAY
         MVC   DIR_READ_BUFFER_LENGTH,=AL4(DIR_READ_BUFFER_LEN)
         XC    PRIMARY_ALET,PRIMARY_ALET PRIMARY ADDRESS SPACE
INVOKE_DIR_READ EQU *
         CALL  BPX1RDD,                READDIR                         X
               (DIR_DESCP,             DIRECTORY FILE DESCRIPTOR       X
               DIRREAD_BUFFER@,        BUFFER                          X
               PRIMARY_ALET,           BUFFER ALET                     X
               DIR_READ_BUFFER_LENGTH, BUFFER SIZE                     X
               RETVAL,                 RET VALUE: Ø, -1, ENTRIES READ  X
               RETCODE,                RETURN CODE                     X
               RSNCODE),               REASON CODE                     X
               VL,                                                     X
               MF=(E,CALLAREA)
         ICM   R6,B'1111',RETVAL       TEST  RETVAL
         BL    READ_DIR_ABEND          -1= FAILURE
         BZ    CLOSE_DIR               ALL DIRECTORY ENTRIES RETURNED
         LA    R3,DIR_READ_BUFFER      DIR READ BUFFER ADDRESS
         USING DIRE,R3                 ADDRESSABILITY TO DIRE
PROCESS_DIR_ENTRY EQU *

**********************************************************************
* THE END OF A DIRECTORY IS INDICATED IN ONE OF TWO WAYS:
*
* 1  A RETURN_VALUE OF Ø ENTRIES IS RETURNED.
*
* 2  SOME FILE SYSTEMS MAY RETURN A NULL NAME ENTRY AS THE LAST ENTRY
*    IN THE CALLERS BUFFER. A NULL ENTRY HAS AN ENTRY_LENGTH OF 4
*    AND A NAME_LENGTH OF Ø.
**********************************************************************
         CLC   DIRENTINFO(L'DIRENTLEN+L'DIRENTNAML),=X'ØØØ4ØØØØ'
         BE    CLOSE_DIR               ALL DIRECTORY ENTRIES RETURNED
         LR    R4,R3                   R4-> DIRE
         LA    R4,L'DIRENTLEN+L'DIRENTNAML(Ø,Ø) START OF NAME
         SLR   R5,R5                   ZEROISE
         ICM   R5,B'ØØ11',DIRENTNAML   LOAD NAME LENGTH
         ALR   R4,R5                   R4->END OF NAME + 1
         USING DIRENTPFSDATA,R4        ADDRESS PHYSICAL FILE SYSTEM
*                                      SPECIFIC DATA
         ICM   R5,B'ØØ11',DIRENTLEN    ENTRY LENGTH
         ALR   R3,R5                   R3-> NEXT DIRE IN BUFFER
         BCT   R6,PROCESS_DIR_ENTRY    PROCESS ALL DIRECTORY ENTRIES
         B     INVOKE_DIR_READ         READ NEXT DIRE
CLOSE_DIR EQU *
         CALL  BPX1CLD,                CLOSEDIR                        X
```

```
               (DIR_DESCP,             DIRECTORY FILE DESCRIPTOR       X
               RETVAL,                 RETURN VALUE: Ø OR -1           X
               RETCODE,                RETURN CODE                     X
               RSNCODE),               REASON CODE                     X
               VL,                                                     X
               MF=(E,CALLAREA)
         ICM   R15,B'1111',RETVAL      CLOSE DIR OKAY?
         BL    CLOSE_DIR_ABEND         BRANCH IF NEGATIVE (-1= FAILURE)
         BR    R2                      RETURN TO CALLER
OPEN_DIR_ABEND EQU  *
         ABEND ØØ1,DUMP                OPEN DIRECTORY ABEND
CLOSE_DIR_ABEND EQU *
         ABEND ØØ2,DUMP                CLOSE DIRECTORY ABEND
READ_DIR_ABEND EQU *
         ABEND ØØ3,DUMP                READ DIRECTORY ABEND
         TITLE ‘STORAGE ITEMS'
CALLL    CALL  ,(,,,,,,,),MF=L
CALLLEN  EQU  *-CALLL                  LENGTH
WORKAREA DSECT
SAVEAREA DS    CL72                    SAVEAREA
PREVSA   EQU   SAVEAREA+4,4            @ OF PREVIOUS SAVEAREA
ROOT     DC    C'/'                    ROOT DIRECTORY
RETVAL   DS    AL4                     RETURN VALUE
RETCODE  DS    AL4                     RETURN CODE
RSNCODE  DS    AL4                     REASON CODE
DIR_DESCP DS   AL4                     DIRECTORY DESCRIPTOR
PRIMARY_ALET   DS AL4                  PRIMARY ALET
SMODE    DS    XL(S_MODE#LENGTH)       FILE MODE FOR MKDIR
DIRREAD_BUFFER@ DS AL4                 DIRECTORY BUFFER ADDRESS
DIR_OPEN_BUFFER DS ØX                  DIRECTORY BUFFER FOR OPEN
DIR_LEN  DS  AL4                       DIRECTORY LENGTH
DIR_NAME DS  CL1Ø23                    DIRECTORY NAME
DIR_OPEN_LEN EQU *-DIR_OPEN_BUFFER     OPEN DIR BUFFER LENGTH
DIR_READ_BUFFER_AREA DS ØX             DIRECTORY READ BUFFER
DIR_READ_BUFFER_LENGTH DS XL4          DIRECTORY BUFFER LENGTH
DIR_READ_BUFFER DS CL4Ø                DIRECTORY BUFFER FOR READ
DIR_READ_BUFFER_LEN EQU *-DIR_READ_BUFFER OPEN DIR BUFFER LENGTH
CALLAREA DS  CL(CALLLEN)               PARM LIST AREA
WORKALEN EQU   *-WORKAREA              WORK AREA LENGTH
         TITLE ‘MAPPING OF A DIRECTORY ENTRY'
         BPXYDIRE DSECT=YES,LIST=YES

***********************************************************************
* THIS EXAMPLE WILL CHANGE THE MODE OF A DIRECTORY                    *
* SERVICES : CHMOD                                                    *
***********************************************************************

MOD_A_DIR EQU *
         MVC   CALLAREA(CALLLEN),CALLL MAKE RENT
         MVC   CHMOD_BUF(L'NEW_DIR),NEW_DIR MODIFY THE DIRECTORY
         MVC   CHMOD_LEN,=AL4(L'NEW_DIR) DIRECTORY NAME LENGTH
         LA    R4,SMODE                FILE MODE AREA
         USING S_MODE,R4               ADDRESSABILITY
         XC    S_MODE,S_MODE           CLEAR MODE FLAGS
***************************
* OWNER= READ/WRITE/SEARCH*
* GROUP= READ/WRITE/WRITE *
* OTHER= READ/SEARCH      *
* MODE = 775              *
***************************
         MVI   S_MODE2,S_IRUSR         775(RWX-RWX-R-X)
         MVI   S_MODE3,S_IWUSR+S_IXUSR+S_IRWXG+S_IROTH+S_IXOTH
         CALL  BPX1CHM,                CHANGE FILE MODES              X
               (CHMOD_LEN,             PATHNAME  LENGTH               X
               CHMOD_BUF,              PATHNAME                       X
               S_MODE,                 BPXYMODE AND BPXYFTYP          X
               RETVAL,                 RETURN VALUE: Ø OR -1          X
               RETCODE,                RETURN CODE                    X
               RSNCODE),               REASON CODE                    X
               VL,                                                    X
               MF=(E,CALLAREA)
         ICM   R4,B'1111',RETVAL       USER INFO RETURNED?
         BL    CHMOD_ABEND             BRANCH IF NEGATIVE (-1= FAILURE)
         BR    R2                      RETURN TO CALLER
CHMOD_ABEND    EQU *
         ABEND ØØ4,DUMP                CHMOD ABEND
         TITLE ‘LITERAL POOL'
         LTORG
         TITLE ‘STORAGE ITEMS'
CALLL    CALL  ,(,,,,,,,),MF=L
CALLLEN  EQU  *-CALLL                  LENGTH
NEW_DIR  DC    CL18'/u/remØØ1/testdir/' mkdir
WORKAREA DSECT
SAVEAREA DS    CL72                    SAVEAREA
PREVSA   EQU   SAVEAREA+4,4            @ OF PREVIOUS SAVEAREA
RETVAL   DS    AL4                     RETURN VALUE
RETCODE  DS    AL4                     RETURN CODE
RSNCODE  DS    AL4                     REASON CODE
SMODE    DS    XL(S_MODE#LENGTH)       FILE MODE FOR CHMOD
CHMOD_BUFFER DS ØX                     CHMOD BUFFER
CHMOD_LEN    DS  AL4                   CHMOD LENGTH
CHMOD_BUF    DS  CL1ØØ                 DIRECTORY NAME MAX 1ØØ CHARS
CALLAREA DS  CL(CALLLEN)               PARM LIST AREA
WORKALEN EQU   *-WORKAREA              WORK AREA LENGTH
         TITLE ‘MAPPING OF A DIRECTORY ENTRY'
         BPXYDIRE DSECT=YES,LIST=YES
         TITLE ‘FILE TYPE DEFINITIONS'
         BPXYFTYP DSECT=MEANINGLESS,LIST=YES
         TITLE ‘MODE CONSTANTS FOR SYSCALL'
         BPXYMODE DSECT=YES,LIST=YES
         TITLE ‘SYSCALL CONSTANTS'
         BPXYCONS DSECT=MEANINGLESS,LIST=YES
```

Rem Perretta
Systems Programmer (UK)

Selecting messages from the log

INTRODUCTION

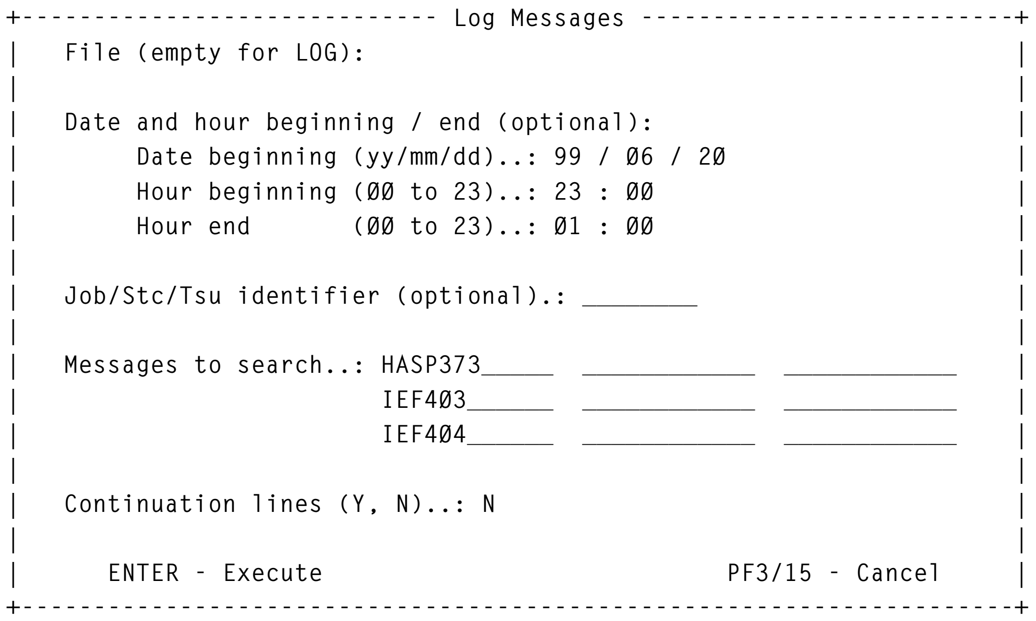
The following program was designed to extract messages from an MVS log or from a log-archiving file. You can select a specific period (a start time and an end time), the message IDs desired (up to nine can be specified), and the task that originated them. All these possibilities are optional.

The main difference between this program and a regular sort is that you can retrieve both the lines that contain the selected messages, and also the continuation lines. In a log there are some messages that span across several lines. I call these continuation lines, and you may wish to retrieve them also, as an option. Furthermore, there are message IDs that appear preceded by some symbols (for example, DFH messages appear with a plus sign before them). For a regular sort, you would need to know this and adjust your sort specifications. This program takes care of that, by automatically adjusting the comparison whenever the message IDs in the log begin with a '+', '-', or '$'. The result of your request is placed in a file, with the same format as a regular log.

This application consists of a REXX EXEC with an associated panel that generates a job. This job copies the actual log to a temporary file, or uses as input an already existing file that you specify. Then it runs an Assembler program that makes the selections and produces the output file. At the beginning of this EXEC there are a few variables that you will need to set, namely the LOADLIB containing the Assembler program module, and the default output filename created. The input panel is shown in Figure 1.

In the example, you want to retrieve messages starting by HASP373, IEF403, and IEF404 that occurred between 23:00 and 1:00 hours, and from no job or STC in particular, since that field is blank. Note that message IDs need not be full message IDs, just the beginning letters. For example, you could specify IEF, and you would get all messages beginning with those letters. The same is true for the job / STC field. This field refers to the job identifier as it appears in log column 38. Also note that we are not asking for continuation lines, and that we want to retrieve the messages from the ‘real’ log, not from some back-up archive, since we leave the ‘File’ field empty.

You can also bypass the entry panel and the EXEC and work directly with a job. To do so, have a look at the job generated by the EXEC. All the DDnames and parameters needed by the Assembler program are explained in the comment on top of it. This way, you can use this program for automated tasks without the need of manual intervention.



```
+---------------------------- Log Messages -------------------------+
|   File (empty for LOG):                                           |
|                                                                   |
|   Date and hour beginning / end (optional):                       |
|        Date beginning (yy/mm/dd)..: 99 / 06 / 20                  |
|        Hour beginning (00 to 23)..: 23 : 00                       |
|        Hour end       (00 to 23)..: 01 : 00                       |
|                                                                   |
|   Job/Stc/Tsu identifier (optional).: ________                    |
|                                                                   |
|   Messages to search..: HASP373_____  ____________  ____________  |
|                         IEF403______  ____________  ____________  |
|                         IEF404______  ____________  ____________  |
|                                                                   |
|   Continuation lines (Y, N)..: N                                  |
|                                                                   |
|      ENTER - Execute                            PF3/15 - Cancel   |
+-------------------------------------------------------------------+
```

Figure 1: The input panel

LOGMESS SOURCE

```
/* REXX MVS *=====================================================*/
/* LOGMESS is a routine to extract selected messages from the log  */
/*         or from a log-like file. Its components are:            */
/*                                                                 */
/*  LOGMESS  - EXEC to generate a job to run the Assembler program.*/
/*  LOGMESSP - An ISPF panel to input data for this EXEC.          */
/*  LOGMESSB - The Assembler program that does the work.           */
/*===============================================================*/
loadlib  = "MY.LOADLIB"               /* where logmessb module is */
tempfile = userid()".TEMPFIL"         /*      temporary file name */
jobnome  = userid()".JOBTEMP"         /*  temporary job file name */
                                      /*         output file name */
outfile  = userid()".LOGMESS.D"right(date("S"),6)".T"time("S")
```

```
do k = 1 to 9
   interpret "M"k"=____________"
end
jobid="________"
c = "N"                                /* default no continuation  */
do forever
   ADDRESS ISPEXEC "ADDPOP ROW(1) COLUMN(1)"
   ADDRESS ISPEXEC "DISPLAY PANEL(LOGMESSP)"
   if rc<>0 then exit
   ADDRESS ISPEXEC "REMPOP"
   if yy="" & mm="" & dd="" & hi="" & hf="" & mi="" & mf="" then leave
   if yy="" | mm="" | dd="" | hi="" | hf="" then do
      ERRO = "Please specify start and end date/time fully"
      iterate
   end
   if mi="" then mi = "00"
   if mf="" then mf = "00"
   if ¬(datatype(yy,"W")&datatype(mm,"W")&,
        datatype(dd,"W")&datatype(hi,"W")&,
        datatype(hf,"W")&datatype(mi,"W")&,
        datatype(mf,"W")) then do
      ERRO = "Invalid non numeric value in date or time"
      iterate
   end
   if mi > 59 | mf > 59 then do
      ERRO = "Invalid minutes specified"
      iterate
   end
   if hi > 24 | hf > 24 then do
      ERRO = "Invalid hour specified"
      iterate
   end
   if mm > 12 | dd > 31 then do
      ERRO = "Invalid month or day specified"
      iterate
   end
   leave
end
continuation = c
datinij = ""
datendj = ""
if yy<>"" then do
   yy = right(yy,2,"0")
   mm = right(mm,2,"0")
   dd = right(dd,2,"0")
   datini  = mm"/"dd"/"yy
   datinij = data_jul(yy||mm||dd)
   if hf < hi then datendj = datinij+1
   else datendj = datinij
   hi = right(hi,2,"0")":"right(mi,2,"0")
   hf = right(hf,2,"0")":"right(mf,2,"0")
end
```

```
if hi = "" then hi = "*"
if hf = "" then hf = "*"
beg_time = strip(datinij hi)
end_time = strip(datendj hf)
x = 0
do k = 1 to 9
   interpret "mesg=M"k
   mesg = space(translate(mesg," ","_"),0)
   if mesg <>"" then do
      zz= left(mesg,1)
      if zz="+"|zz="-"|zz='$' then mesg=substr(mesg,2)
      x = x+1
      msg.x = mesg
   end
end
jobid = space(translate(jobid," ","_"),0)
if jobid = "" then jobid = "*"
xx = msg(off)
delete jobnome
"free  dd (jobe)"
"alloc da('"jobnome"') dd(jobe),
   new blksize(8000) lrecl(80) recfm(f,b),
   dsorg(ps) space(1 1) tracks delete "
if rc<>0 then do
   say "Error "rc" allocating "jobnome
   exit
end
dropbuf
queue "//"userid()"1 JOB CLASS=X,MSGCLASS=X,"
queue "//         MSGLEVEL=(1,1),REGION=2000K"
queue "//*"
queue "//STEP0    EXEC PGM=IEFBR14"
if logfile = "" then do
  queue "//FICTEMP  DD DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,"
  queue "//         DSN="tempfile","
  queue "//         RECFM=FB,LRECL=133,BLKSIZE=13300,"
  queue "//         DSORG=PS,SPACE=(TRK,(90,90))"
end
queue "//SYSPRINT DD SYSOUT=*"
queue "//*"
if logfile = "" then do
  queue "//STEP1    EXEC PGM=SDSF,PARM='++60,228'"
  queue "//ISFOUT   DD DUMMY"
  queue "//ISFIN    DD *"
  queue "LOG"
  queue "PRINT ODSN '"tempfile"' * SHR"
  queue "PRINT 1 999999"
  queue "PRINT CLOSE"
  queue "/*"
  queue "//*"
end
queue "//STEP2    EXEC PGM=LOGMESSB"
if logfile = "" then ,
  queue "//FICTEMP  DD DISP=(OLD,DELETE),DSN="tempfile
else ,
  queue "//FICTEMP  DD DISP=SHR,DSN="logfile
queue "//STEPLIB  DD DISP=SHR,DSN="loadlib
queue "//SYSPRINT DD SYSOUT=*"
queue "//SAIDA    DD DISP=(NEW,CATLG,DELETE),"
queue "//         SPACE=(TRK,(30,30),RLSE),"
queue "//         RECFM=FB,LRECL=133,UNIT=SYSDA,"
queue "//         DSN="outfile
queue "//PARMLINE DD *"
queue continuation
queue beg_time
queue end_time
queue jobid
do j = 1 to x
  queue msg.j
end
queue "/*"
queue ""
"execio * diskw jobe (finis"
"submit '"jobnome"'"
say "Job submitted; See output in "outfile
"free dd(jobe)"
exit
/*================================================================*/
/*        data_jul converts dates from YYMMDD to YYDDD            */
/*================================================================*/
data_jul: procedure
parse arg date_in
aa = left(date_in,2)
mm = substr(date_in,3,2)
dd = right(date_in,2)
if aa//4 = 0 then ac = 1
else ac = 0
select
   when mm = 1   then x = 0
   when mm = 2   then x = 31
   when mm = 3   then x = 59  + ac
   when mm = 4   then x = 90  + ac
   when mm = 5   then x = 120 + ac
   when mm = 6   then x = 151 + ac
   when mm = 7   then x = 181 + ac
   when mm = 8   then x = 212 + ac
   when mm = 9   then x = 243 + ac
   when mm = 10  then x = 273 + ac
   when mm = 11  then x = 304 + ac
   when mm = 12  then x = 334 + ac
   otherwise nop
end
j = x + dd
return aa||right(j,3,'0')
```

LOGMESSP SOURCE

```
)ATTR
  _ TYPE(INPUT)  COLOR(RED)    JUST(LEFT)  CAPS(ON)
  $ TYPE(OUTPUT) COLOR(WHITE)  SKIP(ON)    INTENS(HIGH)
  ? TYPE(TEXT)   COLOR(PINK)   SKIP(ON)    INTENS(HIGH)
  % TYPE(TEXT)   COLOR(YELLOW) SKIP(ON)    INTENS(HIGH)
  # TYPE(TEXT)   COLOR(WHITE)  SKIP(ON)    INTENS(HIGH)
  + TYPE(TEXT)   COLOR(GREEN)  SKIP(ON)    INTENS(LOW)
)BODY WINDOW(71,19)
+
?  File (empty for LOG):_LOGFILE                                      +
+
?  Date and hour beginning / end (optional):
%       Date beginning (yy/mm/dd)..:_YY%/_MM%/_DD+
%       Hour beginning (00 to 23)..:_HI%:_MI+
%       Hour end       (00 to 23)..:_HF%:_MF+
+
?  Job/Stc/Tsu identifier (optional).:_JOBID   +
+
?  Messages to search..:_M1          +_M2          +_M3          +
%                       _M4          +_M5          +_M6          +
%                       _M7          +_M8          +_M9          +
+
?  Continuation lines (Y, N)..:_C
+
#       $ERRO
|     ENTER - Execute                            PF3/15 - Cancel
)INIT
&END = PFK(END)
&ZWINTTL = 'Log Messages'
)END
```

LOGMESSB SOURCE

```
*===================================================================*
*                                                                   *
* LOGMESSB - This program extracts messages from a log-type file,   *
*            including continuation lines. The input file must be   *
* identical to the produced by an SDSF PRINT command (with CCs      *
* at column one). The following DDnames are used in this program:  *
*                                                                   *
* FICTEMP  - The input file containing a log print.                 *
* SAIDA    - The output file, similar to the input, containing      *
*            only the desired messages.                             *
* SYSPRINT - Standard job output.                                   *
* PARMLINE - Parameter file that controls message selection.        *
*            It's format is as follows:                             *
* Line1.:C            Continuation lines (Y or N).                  *
* Line2.:YYDDD HH:MM  Start date/hour or * for all.                 *
* Line3.:YYDDD HH:MM  End   date/hour or * for all.                 *
* Line4.:Job/Stc/Tsu as it appears at log column 38 or * for all.   *
* Line5 onwards: message identifier as it appears at log column 57. *
*     Up to 9 identifiers can be specified, one per line, or none.  *
*     Do not include  + - $  symbols that appear at the beginning   *
*     of some messages.                                             *
*                                                                   *
*===================================================================*
&PROGRAM SETC  'LOGMESSB'          This program's name
&JOBPOS  SETC  '38'                Jobid   column in log file
&NUMPOS  SETC  '43'                Number  column in log file
&MSGPOS  SETC  '57'                Message column in log file
&DATPOS  SETC  '2Ø'                Date    column in log file
&TL      SETC  '12'                Length of parm table entries
&PROGRAM CSECT                     (minus 4 bytes for length).
&PROGRAM AMODE 31
&PROGRAM RMODE 24
         SAVE  (14,12)             Start stuff
         LR    R12,R15
         USING &PROGRAM,R12
         ST    R13,SAVEA+4
         LA    R11,SAVEA
         ST    R11,8(R13)
         LR    R13,R11
         B     OPENPRT
         DC    CL16' &PROGRAM 1.1'
         DC    CL8'&SYSDATE'
*                                  Open files
OPENPRT  DS    ØH
         OPEN  (SYSPRINT,OUTPUT)
         LTR   R15,R15
         BNZ   EXIT
OPENSAI  DS    ØH
         OPEN  (SAIDA,OUTPUT)
         LTR   R15,R15
         BZ    OPENTEMP
         MVC   XMSGTYPE,=C'OPEN '
         MVC   XMSGDSN,=CL44'SAIDA. Program terminated.'
         PUT   SYSPRINT,XMSGLINE
         B     EXIT

OPENTEMP DS    ØH
         OPEN  (FICTEMP,INPUT)
         LTR   R15,R15
         BZ    OPENPARM
         MVC   XMSGTYPE,=C'OPEN '
         MVC   XMSGDSN,=CL44'TEMPFILE. Program terminated.'
         PUT   SYSPRINT,XMSGLINE
         B     EXIT
OPENPARM DS    ØH
         OPEN  (PARMLINE,INPUT)
         LTR   R15,R15
         BZ    GETCONTN
         MVC   XMSGTYPE,=C'OPEN '
         MVC   XMSGDSN,=CL44'PARMLINE. Program terminated.'
         PUT   SYSPRINT,XMSGLINE
         B     EXIT
*===================================================================*
*  Read parameter file and process it.                              *
*===================================================================*
GETCONTN EQU   *                    Get continuation lines option
         GET   PARMLINE,CONTINUE
GETBEGTI EQU   *                    Get beginning time
         L     R6,=F'-1'            Assume no begtime specified.
         L     R7,=F'-1'            Assume no endtime specified.
         L     R2,=F'-1'            Assume no job/stc specified.
         XR    R5,R5                R5 is table elements counter
         LA    R4,MSGTAB            Load message table address in R4
         GET   PARMLINE,BEGTIME     Get beginning date and hour
         CLI   BEGTIME,C'*'         No begtime?
         BE    GETENDTI             No, jump ahead.
         L     R6,=F'1Ø'            Otherwise, leng is 11 (Ex 1Ø).
GETENDTI EQU   *
         GET   PARMLINE,ENDTIME     Same thing for endtime.
         CLI   ENDTIME,C'*'         No endtime?
         BE    GETJOBID             No, jump ahead.
         L     R7,=F'1Ø'            Otherwise, leng is 11 (Ex 1Ø).
GETJOBID EQU   *
         GET   PARMLINE,JOBID       Get job selection
         CLI   JOBID,C'*'           All jobs?
         BE    GETPARM              Yes, jump
         SR    R9,R9                No, find jobname length
         LA    R3,JOBID             Copy address to R3
         BAL   R1Ø,FINDSPC          Call find space subroutine
         SH    R9,=H'1'             Length ready for executed CLC
         LR    R2,R9                Keep it in R2
GETPARM  EQU   *
         GET   PARMLINE,Ø(Ø,R4)     Get parms (msg ids)
         SR    R9,R9                Parm table entry
         LR    R3,R4                Copy initial pointer to R3
         BAL   R1Ø,FINDSPC          Call find space subroutine
         SH    R9,=H'1'             Length ready for executed CLC
         ST    R9,&TL+Ø(Ø,R4)       Store length after parm
         LA    R4,&TL+4(Ø,R4)       Next tab entry (TL + 4)
         LA    R5,1(Ø,R5)           Increment counter
         CH    R5,=H'9'             Limit of 9 attained?
         BE    PARMEND              Yes, ignore others and go ahead
         B     GETPARM              Otherwise, get another.
PARMEND  EQU   *
         CLOSE PARMLINE
         ST    R5,MSGNUM            Store number of table elements
         XR    R8,R8                Clear lines read counter
         MVI   LMESSAGE,C'Ø'        Clear message flag
```

```
*====================================================================*
*  Read a line from the log file and process it.                     *
*====================================================================*
READFILE EQU   *
         GET   FICTEMP,WFICTEMP     Read log line
         CLI   WFICTEMP,C'1'        Header line with CC on column 1
         BE    READFILE             Yes, ignore it
         LA    R8,1(Ø,R8)           Increment line counter
         CLI   CONTINUE,C'N'        No continuation lines?
         BNE   READFIL1             Yes, jump ahead
         CLC   &DATPOS+WFICTEMP(23),=23C' '  Date, time or job/stc?
         BE    READFILE             Yes, read next
         B     BEGTEST              Else, jump to test start date.
*
READFIL1 EQU   *
         CLI   LMESSAGE,C'Ø'        Message flag on?
         BZ    SEEKNUM              No, jump ahead
         CLC   &DATPOS+WFICTEMP(26),=26C' '  Date, time or job/stc?
         BNE   SEEKNUM              Yes, jump
         PUT   SAIDA,WFICTEMP       Else, write line (type 1 line)
         B     FINDNUMB             Go look for continuation number.
*====================================================================*
*  Check for a "continuation number" in numpos (line type 2).        *
*  If so, search for the number in numtable. If found, write line.  *
*====================================================================*
SEEKNUM  EQU   *                    Continuation number process
         CLC   NUMTOTAL,=H'Ø'       Are there elements in numtab?
         BE    BEGTEST              No, jump ahead.
         CLC   &JOBPOS+WFICTEMP(5),=5C' ' Jobpos first 5 bytes blank?
         BNE   BEGTEST              No, jump ahead
         CLC   &NUMPOS+WFICTEMP(3),=3C' ' Numpos 3 bytes blank?
         BE    BEGTEST              Yes, jump ahead
         LA    R9,NUMTABLE          Load numbers table address.
SEEKNUM1 EQU   *
         CLC   Ø(2,R9),=C'  '       Entry erased (2 spaces),
         BE    SEEKNEXT             loop to next.
         CLI   Ø(R9),X'ØØ'          Logical end of table(lowvalue),
         BE    BEGTEST              exit search.
         CLI   Ø(R9),X'FF'          Physical end of table(highvalue),
         BE    BEGTEST              exit search.
SEEKNUM2 EQU   *
         LR    RØ,R8                Compare current line (R8) minus
         S     RØ,=F'1ØØ'           1ØØ with line number in table.
         C     RØ,4(Ø,R9)           If greater, call routine to
         BL    SEEKNUM3             clear table entry.
         BAL   R11,CLEARTAB
         B     SEEKNEXT
SEEKNUM3 EQU   *
         CLC   1(3,R9),&NUMPOS+WFICTEMP Compare number
         BNE   SEEKNEXT             No match, loop to next entry.
         PUT   SAIDA,WFICTEMP       Match, write line (type 2)
         CLC   WFICTEMP(2),=C' E'   End multiple line?
         BNE   READFILE             No, read next
         BAL   R11,CLEARTAB         Yes, clear table entry
         B     READFILE             and read next.
SEEKNEXT EQU   *
         LA    R9,8(Ø,R9)           Next entry (8 bytes more)
         B     SEEKNUM1
*====================================================================*
*  Test for start / end hour and date.                               *
*====================================================================*
BEGTEST  EQU   *                    Start date hour test
         CLC   WFICTEMP(2),=C' S'   Sequence (continuation) line?
         BE    READFILE             Yes, read next
         MVI   LMESSAGE,C'Ø'        Reset message flag
         LTR   R6,R6                Start specified?
         BM    ENDTEST              If R6 negative, no, jump
         EX    R6,COMPARE2          Else, compare
         BL    READFILE             Lower: start not reached
         L     R6,=F'-1'            Start reached: destroy test
         B     JOBTEST              Else, accept line.
ENDTEST  EQU   *                    End date hour test
         LTR   R7,R7                End specified?
         BM    JOBTEST              If R7 negative, no, jump
         EX    R7,COMPARE3          Else, compare
         BH    EXIT                 High: end reached, exit program.
*====================================================================*
*  Test for jobnames / stcnames and for message ids.                 *
*====================================================================*
JOBTEST  EQU   *
         LTR   R2,R2                Job/stc/tsu id specified?
         BM    MSGTEST              No, jump ahead (R2 negative)
         EX    R2,COMPARE4          Yes, execute compare
         BNE   READFILE             No match, read another line.
MSGTEST  EQU   *
         L     R11,MSGNUM           Number of msg ids specified.
         LTR   R11,R11              If zero, write all lines,
         BZ    WRITLINØ             otherwise, load msg table
         LA    R4,MSGTAB            address for compares.
MSGTEST1 EQU   *                    See if message match parms
         L     R5,&TL+Ø(Ø,R4)       Load parm length (EX ready)
         CLI   &MSGPOS+WFICTEMP,C'+' Message starts by '+'?
         BE    MSGTEST2             Yes, jump to correct execute
         CLI   &MSGPOS+WFICTEMP,C'-' Message starts by '-'?
         BE    MSGTEST2             Yes, jump to correct execute
         CLI   &MSGPOS+WFICTEMP,C'$' Message starts by '$'?
         BE    MSGTEST2             Yes, jump to correct execute
         EX    R5,COMPAREØ          Compare message
         BNE   MSGTEST3             Not equal, loop for next
         B     WRITLINØ             Otherwise, junp to write line
MSGTEST2 EQU   *                    Compare one byte ahead for
         EX    R5,COMPARE1          messages starting with + - $.
         BE    WRITLINØ
MSGTEST3 EQU   *
         LA    R4,&TL+4(Ø,R4)       Next table address (4+TL)
         BCT   R11,MSGTEST1         Loop for number of entries.
         B     READFILE
WRITLINØ EQU   *
         PUT   SAIDA,WFICTEMP       Write line (type Ø line).
         CLI   CONTINUE,C'N'        No continuation lines?
         BE    READFILE             No, read next line.
*====================================================================*
*  See if a line ends with a three-digit number.                     *
*====================================================================*
FINDNUMB EQU   *
         LA    R1Ø,WFICTEMP+132     Point to end of line
FINDNUM  EQU   *
         CLI   Ø(R1Ø),X'4Ø'         Look for three digits
         BNE   FINDNUM1             preceded by a space.
         BCT   R1Ø,FINDNUM          Loop to beginning of line.
         B     FINDNUMF             Empty line?
FINDNUM1 EQU   *
         CLI   Ø(R1Ø),C'Ø'
         BL    FINDNUMF
         CLI   Ø(R1Ø),C'9'
         BH    FINDNUMF
         S     R1Ø,=F'1'
         CLI   Ø(R1Ø),C'Ø'
         BL    FINDNUMF
         CLI   Ø(R1Ø),C'9'
         BH    FINDNUMF
         S     R1Ø,=F'1'
         CLI   Ø(R1Ø),C'Ø'
         BL    FINDNUMF
         CLI   Ø(R1Ø),C'9'
         BH    FINDNUMF
         S     R1Ø,=F'1'
         CLI   Ø(R1Ø),X'4Ø'
         BE    SEEKFREE             3 digits found, move ahead.
FINDNUMF EQU   *                    Not 3 digits, set message flag
         MVI   LMESSAGE,C'1'        and read next line.
         B     READFILE
*====================================================================*
*  Seek for a free entry in numtable to store the number.            *
*  In the way, clear old entries (over 1ØØ lines gone by).           *
*====================================================================*
SEEKFREE EQU   *                    Seek for a free entry in the table
         LA    R9,NUMTABLE          R9 points beginning of table
SEEKLOOP EQU   *                    Look for a free entry in the
         CLC   Ø(2,R9),=C'  '       table (either with spaces or
         BE    MOVENUM              low-values).
         CLI   Ø(R9),X'ØØ'
         BE    MOVENUM
         LR    RØ,R8                Compare current line (R8) minus
         S     RØ,=F'1ØØ'           1ØØ with line number in table.
         C     RØ,4(Ø,R9)           If greater, clear table entry.
         BL    SEEKFRE1
         BAL   R11,CLEARTAB
SEEKFRE1 EQU   *
         LA    R9,8(Ø,R9)           Increment tab pointer
         CLI   Ø(R9),X'FF'          End of table?
         BE    READFILE             Yes, abandon the search.
         B     SEEKLOOP             Else loop.
MOVENUM  EQU   *                    Found empty entry, move number
         MVC   Ø(4,R9),Ø(R1Ø)       (with initial space).
         ST    R8,4(Ø,R9)           Store line number.
         LH    RØ,NUMTOTAL          Increment numtotal.
         AH    RØ,=H'1'
         STH   RØ,NUMTOTAL
         B     READFILE
*====================================================================*
*  Close files and exit.                                             *
*====================================================================*
EXIT     EQU   *
         CLOSE SAIDA
         CLOSE FICTEMP
         CLOSE SYSPRINT
         L     R13,SAVEA+4
         LM    R14,R12,12(R13)
         XR    R15,R15
         BR    R14
*====================================================================*
*  Subroutines, executed instructions and work areas.                *
*====================================================================*
FINDSPC  EQU   *                    Find space subroutine: counts the
         CLI   Ø(R3),X'4Ø'          number of chars in a string up to
         BE    FINDSPCF             the first space.
         LA    R9,1(Ø,R9)           Returns the number in R9.
         LA    R3,1(Ø,R3)           R3 points current position
         B     FINDSPC
FINDSPCF EQU   *
         BR    R1Ø                  Return
*
CLEARTAB EQU   *                    Clear table entry with spaces
         MVC   Ø(8,R9),=8C' '
         LH    RØ,NUMTOTAL          Decrement numtotal.
         SH    RØ,=H'1'
         STH   RØ,NUMTOTAL
         BR    R11                  Return
*
COMPAREØ CLC   &MSGPOS+WFICTEMP(Ø),Ø(R4)   Compare message id
COMPARE1 CLC   &MSGPOS+WFICTEMP+1(Ø),Ø(R4) Same, but 1 byte ahead
COMPARE2 CLC   &DATPOS+WFICTEMP(Ø),BEGTIME Compare start date+hour
COMPARE3 CLC   &DATPOS+WFICTEMP(Ø),ENDTIME Compare end date+hour
```

```
COMPARE4 CLC   &JOBPOS+WFICTEMP(Ø),JOBID   Compare jobnames
*
         LTORG
SAVEA    DS    18F                  Save register area
CONTINUE DS    C                    Continuation lines option
         DS    ØF
BEGTIME  DS    CL12                 Beg date and hour (yyddd hh:mm)
ENDTIME  DS    CL12                 End date and hour.
JOBID    DS    CL8                  Job selection
MSGNUM   DS    F                    Number of entries in parm table
MSGTAB   DS    36F                  9 entries max (12+4 bytes each)
WFICTEMP DS    CL133                Log line read area
LMESSAGE DS    C                    Message flag (zero or one)
NUMTOTAL DC    H'Ø'                 Total entries in numtable
NUMTABLE DC    6ØF'Ø'               Numbers table: 3Ø entries.
ENDTABLE DC    X'FF'                End of table mark.
XMSGLINE DS    ØCL8Ø                Error messages line
         DC    C'===>>>>> Error in '
XMSGTYPE DC    C'     '
         DC    C' dataset '
XMSGDSN  DC    CL5Ø' '
*
PARMLINE DCB   DSORG=PS,RECFM=FB,MACRF=(GM),                           X
               LRECL=8Ø,                                               X
               EODAD=PARMEND,                                          X
               DDNAME=PARMLINE
*
FICTEMP  DCB   DSORG=PS,RECFM=FB,MACRF=(GM),                           X
               LRECL=133,                                              X
               EODAD=EXIT,                                             X
               DDNAME=FICTEMP
*
SAIDA    DCB   DSORG=PS,RECFM=FB,MACRF=(PM),                           X
               LRECL=133,                                              X
               DDNAME=SAIDA
*
SYSPRINT DCB   DSORG=PS,RECFM=FB,MACRF=(PM),                           X
               LRECL=8Ø,                                               X
               DDNAME=SYSPRINT
*
         YREGS
         END
```

Luis Paulo Ribeiro
Systems Engineer
Edinfor (Portugal)

MVS news

BMC has launched Version 2.1 of InTune OS/390 application performance tuner, promising an ability to share information between systems via Parallel Sysplex, to analyse the target application, and simplify customization of batch reports. The product analyses the performance of programs running on OS/390, CICS, DB2, IMS, NATURAL, ADABAS, and Datacom, and pinpoints code efficiency problems.

Users can fine-tune applications in production, in test, or under development. Version 2.1 identifies application program delays and presents this information for analysis through an interactive interface for both traditional and Parallel Sysplex environments. The product lets users go inside the code and view precisely where a resource delay is occurring. In-built support for Parallel Sysplex allows organizations to share performance information between various systems within the sysplex, allowing users to choose all or any specific individual systems when invoking requests.

For further information contact:

BMC Software, Inc, 2101 Citywest Blvd, Houston, TX 77042, USA.
Tel: (713) 918 8800
Fax: (713) 918 8000 or

BMC Software Ltd, Compass House, 207-215 London Road, Camberley, Surrey, GU15 3E, UK.
Tel: (01276) 24622
Fax: (01276) 61201
http://www.bmc.com

* * *

PROIV has announced PROIV Mainframe 4.0, a Web-enabled version of its System/390-based application development software. PROIV Mainframe 4.0 has a full Windows-style GUI, and offers the ability to port applications between OS/390, Unix, and NT.

The utility allows developers to create Windows and Java front-ends for mainframe applications, opening up the whole environment to the Internet and intranet using standard Web browsers. The PROIV JavaSuite allows applications to be delivered to Web browser clients without the need to amend application code.

PROIV applications are database-independent and may be deployed on mainframes running DB2, DL/I or VSAM. PROIV Mainframe 4.0 supports the following clients; Windows 95/98 and NT, 3270, 5550 (double-byte character support), and Web browsers (Java); it is also Year 2000 compliant.

For further information contact:

PROIV Software Inc, 101 Academy, Suite 200, Irvine, California, USA.
Tel: (949) 823 1000
Fax: (949) 823 1010 or

PROIV Ltd, King’s Hall, Parson’s Green, St Ives, Cambridgeshire, PE17 4W7, UK.
Tel: (1480) 494330
Fax: (1480) 494 039
http://www.proiv.com

* * *