55 TCP/SNA update

September 2004

In this issue

TCP/SNA Update

Published by

Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

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

Editor

Trevor Eddolls
E-mail: trevore@xephon.com

Publisher

Nicole Thomas
E-mail: nicole@xephon.com

Subscriptions and back-issues

A year's subscription to *TCP/SNA Update*, comprising four quarterly issues, costs $190.00 in the USA and Canada; £130.00 in the UK; £136.00 in Europe; £142.00 in Australasia and Japan; and £140.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the March 2000 issue, are available separately to subscribers for $49.50 (£33.00) each including postage.

Editorial panel

Articles published in TCP/SNA Update are reviewed by our panel of experts. Members include John Bradley (UK), Carlson Colomb (Canada), Anura Gurugé (USA), Jon Pearkins (Canada), and Tod Yampel (USA).

Contributions

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

TCP/SNA Update on-line

Code from *TCP/SNA Update*, and complete issues in Acrobat PDF format, can be downloaded from http://www.xephon.com/tcpsna; you will need to supply a word from the printed issue..

Printed in England.

A technique for transferring and processing files from a PC to a host

INTRODUCTION

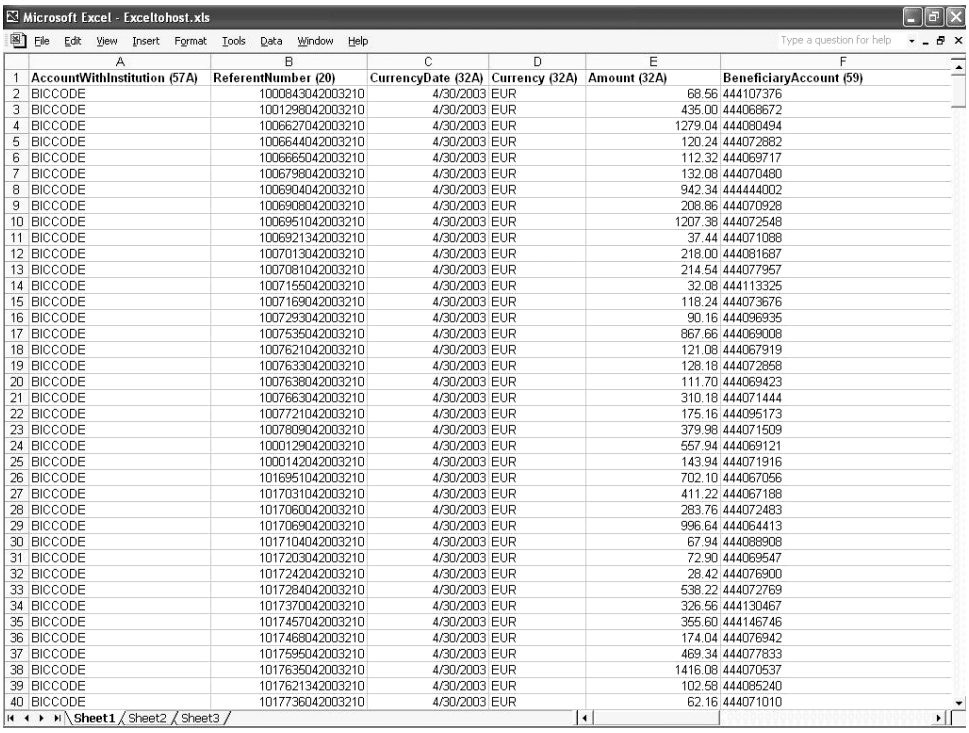
This article describes one technique for file transfer and processing from a PC to a mainframe computer.

Occasionally one needs to transfer a file from a PC to a host computer, and to update files on the mainframe with the data in that file. Users in my company regularly receive, by e-mail, an Excel file with data, in which foreign currency accounts are credited. For that purpose, we have used a CICS transaction requiring the user to type in the account number, currency code, and amount. However, as the number of records in the files increased over time, the amount of manual work (typing) increased to a point where the use of a CICS transaction became completely ineffective. Therefore I decided to create a PC procedure that will transfer a file to the host, start a job with a batch program instead of a CICS transaction, and return its result to the PC. It had to be done in the simplest way possible, preferably by pressing a few buttons, and also with a return message that unambiguously explains any possible problems.

PC SIDE

First of all, it was necessary to create a file with a predefined record description, to be transferred to the host. It was done by formatting the Excel file, named Exceltohost.xls, which looks like Figure 1.

This file is in fact an Excel form of a standard financial message of type 103, arriving via e-mail instead of through the SWIFT network. The fields of interest are BeneficiaryAccount (column F), Currency (D), and Amount (E). A file with these fields could have been created in several ways. I used Visual Basic for Applications as a standard part of Microsoft Excel. It is invoked



Microsoft Excel - Exceltohost.xls

File Edit View Insert Format Tools Data Window Help

	A	B	C	D	E	F
1	AccountWithInstitution (57A)	ReferentNumber (20)	CurrencyDate (32A)	Currency (32A)	Amount (32A)	BeneficiaryAccount (59)
2	BICCODE	1000843042003210	4/30/2003	EUR	68.56	444107376
3	BICCODE	1001298042003210	4/30/2003	EUR	435.00	444068672
4	BICCODE	1006627042003210	4/30/2003	EUR	1279.04	444080494
5	BICCODE	1006644042003210	4/30/2003	EUR	120.24	444072882
6	BICCODE	1006665042003210	4/30/2003	EUR	112.32	444069717
7	BICCODE	1006798042003210	4/30/2003	EUR	132.08	444070480
8	BICCODE	1006904042003210	4/30/2003	EUR	942.34	444444002
9	BICCODE	1006908042003210	4/30/2003	EUR	208.86	444070928
10	BICCODE	1006951042003210	4/30/2003	EUR	1207.38	444072548
11	BICCODE	1006921342003210	4/30/2003	EUR	37.44	444071088
12	BICCODE	1007013042003210	4/30/2003	EUR	218.00	444081687
13	BICCODE	1007081042003210	4/30/2003	EUR	214.54	444077957
14	BICCODE	1007155042003210	4/30/2003	EUR	32.08	444113325
15	BICCODE	1007169042003210	4/30/2003	EUR	118.24	444073676
16	BICCODE	1007293042003210	4/30/2003	EUR	90.16	444096935
17	BICCODE	1007535042003210	4/30/2003	EUR	867.66	444069008
18	BICCODE	1007621042003210	4/30/2003	EUR	121.08	444067919
19	BICCODE	1007633042003210	4/30/2003	EUR	128.18	444072858
20	BICCODE	1007638042003210	4/30/2003	EUR	111.70	444069423
21	BICCODE	1007663042003210	4/30/2003	EUR	310.18	444071444
22	BICCODE	1007721042003210	4/30/2003	EUR	175.16	444095173
23	BICCODE	1007809042003210	4/30/2003	EUR	379.98	444071509
24	BICCODE	1000129042003210	4/30/2003	EUR	557.94	444069121
25	BICCODE	1000142042003210	4/30/2003	EUR	143.94	444071916
26	BICCODE	1016951042003210	4/30/2003	EUR	702.10	444067056
27	BICCODE	1017031042003210	4/30/2003	EUR	411.22	444067188
28	BICCODE	1017060042003210	4/30/2003	EUR	283.76	444072483
29	BICCODE	1017069042003210	4/30/2003	EUR	996.64	444064413
30	BICCODE	1017104042003210	4/30/2003	EUR	67.94	444088908
31	BICCODE	1017203042003210	4/30/2003	EUR	72.90	444069547
32	BICCODE	1017242042003210	4/30/2003	EUR	28.42	444076900
33	BICCODE	1017284042003210	4/30/2003	EUR	538.22	444072769
34	BICCODE	1017370042003210	4/30/2003	EUR	326.56	444130467
35	BICCODE	1017457042003210	4/30/2003	EUR	355.60	444146746
36	BICCODE	1017468042003210	4/30/2003	EUR	174.04	444076942
37	BICCODE	1017595042003210	4/30/2003	EUR	469.34	444077833
38	BICCODE	1017635042003210	4/30/2003	EUR	1416.08	444070537
39	BICCODE	1017621342003210	4/30/2003	EUR	102.58	444085240
40	BICCODE	1017736042003210	4/30/2003	EUR	62.16	444071010

Sheet1 / Sheet2 / Sheet3

Figure 1: Format of the Excel file

very simply by clicking on View on the menu, and then choosing the Visual Basic toolbar – see Figure 2.

Then I added a command button to the Excel file by clicking on the control toolbox, double-clicking on the Command Button option and set its caption to ‘Make File’ – see Figure 3.

By clicking on the Visual Basic Editor button on the VB toolbar the Visual Basic editor appears – see Figure 4.

Then I added code to the CommandButton1_Click procedure that executes every time the button is pressed. Assuming that every new file received by e-mail will be added to Exceltohost.xls, this code prompts for the first and last row of the Excel file that will be processed and makes a file with the specified record description (written in PL/I):

```
DCL 1 record_desc,
      2  account_number PIC'(9)9',
```

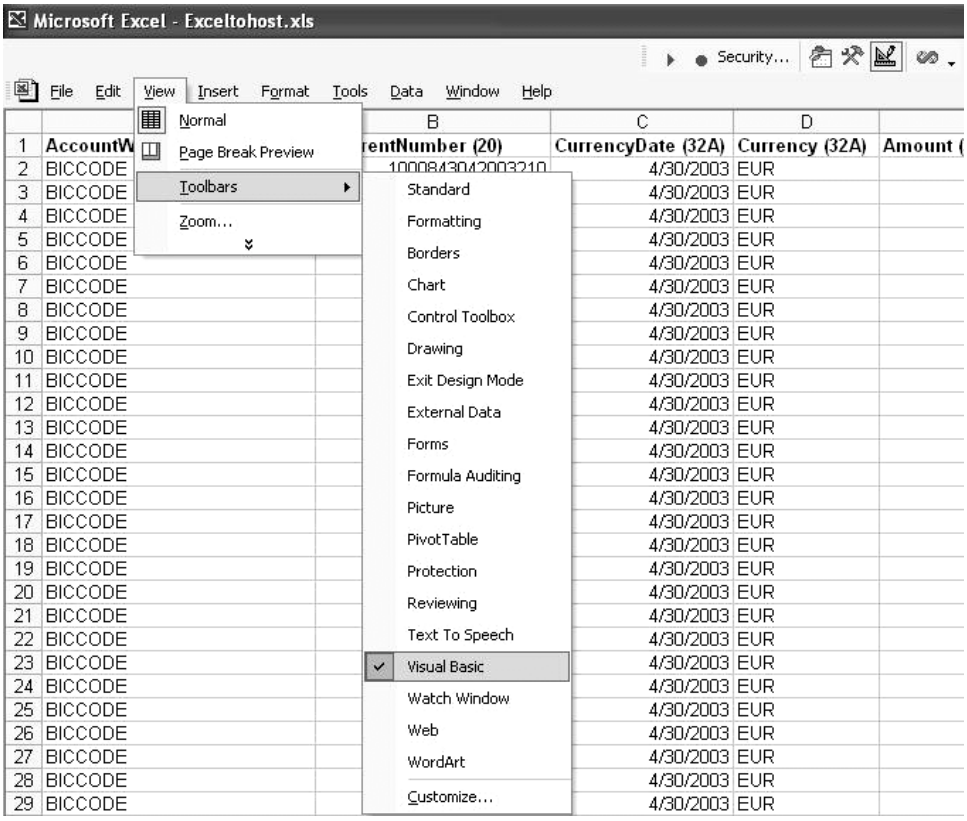


Figure 2: Selecting Visual Basic

```
      2  currency        CHAR(3),
      2  amount          PIC'(5)Z9V.99';
```

This file is renamed to_host.txt in the current_directory.

In case of changes (if the order of the columns is changed, etc), some additional controls should be added. This code could be easily altered. This is an advantage of using VBA.

Here is the code:

```
Private Sub CommandButton1_Click()
    Dim rows, columns, first_row, last_row As Integer
    Dim account_number, amount As String * 9
    Dim currency As String * 3
    Dim input_value, line As String

    On Error GoTo error
```

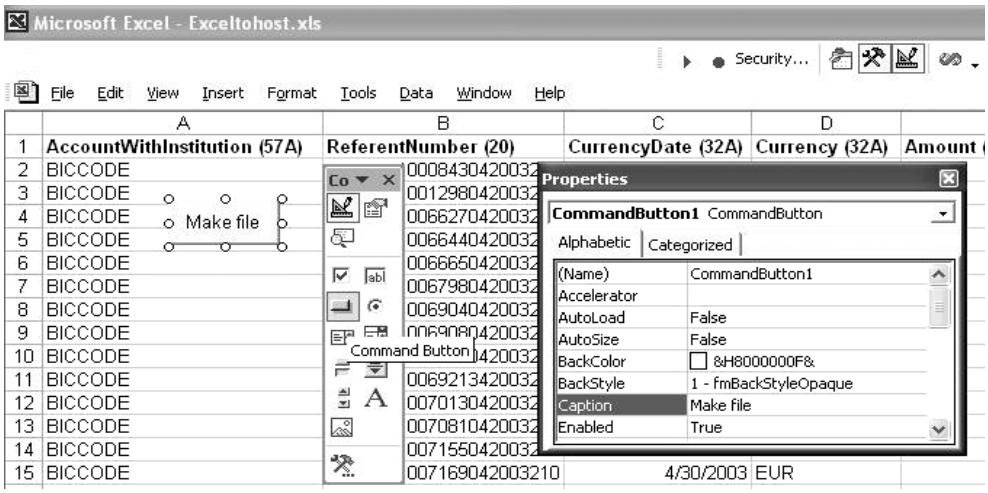


Figure 3: Make file

```
    input_value = InputBox("Enter the number of the first row ", "FIRST
ROW")
    If input_value = "" Or IsNumeric(Val(input_value)) = False Then
        Exit Sub
        Else: first_row = Trim(Val(input_value))
    End If

 input_value = InputBox("Enter the number of the last row ", "LAST ROW")
    If input_value = "" Or IsNumeric(Val(input_value)) = False Then
        Exit Sub
        Else: last_row = Trim(Val(input_value))
    End If

    Open "C:\current_directory\to_host.txt" For Output As #1

    For rows = first_row To last_row
        For columns = 4 To 6
            With Worksheets("Sheet1").Cells(rows, columns)
            Select Case (columns)
                Case 4: currency = CStr(.Value)
                Case 5: RSet amount = CStr(.Value)
                Case 6: account_number = CStr(.Value)
            End Select
            End With
            line = account_number & currency_name & amount
        Next columns
        Print #1, line
    Next rows
```

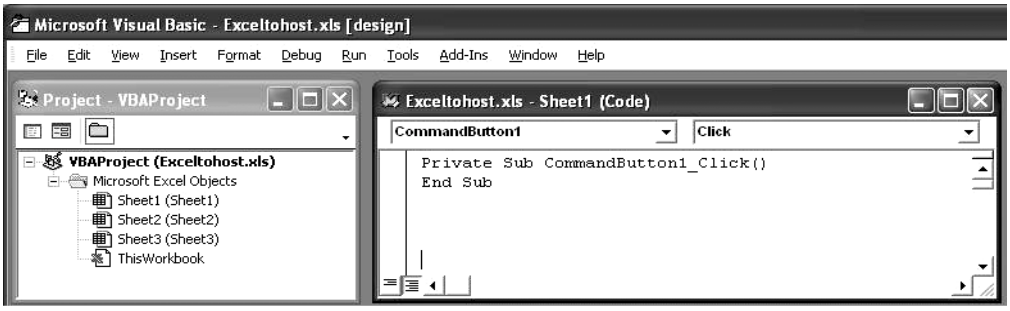


Figure 4: Visual Basic editor

```
    Close #1
    MsgBox "File made, start the transfer"
    Exit Sub
error:
        MsgBox Err.Description
        Exit Sub
End Sub
```

Then I created a batch procedure that:

1 Starts the file transfer protocol.

2 Sends the file to_host.txt to the host, replacing the existing host file, frompc, which has the same record length and description as to_host.txt and enough space to accept the data.

3 Starts job pchost.

4 Returns its result in a file called report_from_host.txt in the current directory.

It is a transfer.bat procedure in the current directory that looks like:

```
FTP -S: transfer.txt
```

which means execute the file transfer protocol commands in transfer.txt. Without –S:, FTP would be started in DOS mode and prompt for each command, which was not the idea.

Transfer.txt looks like this:

```
OPEN host_ip_address
```

Figure 5: Transfer form

```
user
password
CD \current_directory
PUT to_host.txt  'frompc' (REPLACE
CD..
LITERAL SITE FILE=JES
GET job_library(pchost)  C:\current_directory\report_from_host.txt
QUIT
```

The line LITERAL SITE FILE=JES invokes the host TCP/IP SITE subcommand, which, with the FILE=JES option, enables our remote job submission, and LITERAL is used because FTP invokes the IBM host command from a Microsoft platform.

Then I created a Visual Basic program with one simple form, frmTRANSFER, as a convenient graphical interface for starting the transfer.bat procedure and showing its result. This could not be done under VBA, since it does not support shell commands.

The form looks like Figure 5.

It has the following control tools – two command buttons (cmdDO and cmdEXIT), one text box (txtMESSAGE) for showing messages, and a timer (Timer1 – not visible at run-time) set to attempt to read report_from_host.txt every 30 seconds. By pressing *Do*, the transfer is started.

Here is the code:

```
Private Sub cmdOK_Click()
    Kill "C:\current_directory\report_from_host.txt "
rem deletes last report
    retval = Shell("C:\current_directory\transfer.bat", 6)
```

```
    txtMESSAGE.Text = "Transfer started"
    Timer1.Enabled = True
rem after 30 seconds, the answer from host will be checked
    cmdDO.Enabled = False
rem transfer could be started only once !
End Sub

Private Sub Timer1_Timer()
    Dim line As String
    On Error GoTo error

    txtMESSAGE.Text = ""
    Open "C:\current_directory\report_from_host.txt " For Input As #1
    Do While Not EOF(1)
            Line Input #1, line
            txtMESSAGE.Text= txtMESSAGE.Text & " " & line
     Loop
     Close #1
     Timer1.Enabled = False
rem report from host is shown and timer is disabled
    Exit Sub
error:
     If Err.Number = 53 Then
      txtMESSAGE.Text=" No answer yet !"
      Else:  MsgBox Err.Description
     End If
End Sub

Private Sub cmdEXIT_Click()
    Unload frmTRANSFER
    End
End Sub
```

Its executable version is placed in the current directory.

HOST SIDE

On the host side, the job, control job_library(pchost), which is started from the PC, looks like this:

```
//USER JOB MSGCLASS=I,MSGLEVEL=(1,1)
//step001  EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=X
//SYSUDUMP DD SYSOUT=*
//errors   DD SYSOUT=*,DCB=(RECFM=FBA,LRECL=80)
//filein   DD DSN=frompc,DISP=SHR
//SYSTSIN  DD *
   DSN SYSTEM(DSN)
```

```
   RUN  PROGRAM(pchost) PLAN(pchost) -
      LIB('progam_library')
/*
```

The jobname USER matches the user who logged on to the host.

It is important to note that the SYSPRINT file has a different SYSOUT class from the other files. In this particular case MSGCLASS=I means that SYSOUT files will be ‘collected by’ and shown in BETA 92 report manager, except SYSPRINT, which has SYSOUT=X. This way, only those messages that the program pchost produces by the PUT SKIP EDIT command (which go to the SYSPRINT file) will be shown on the PC. This is because users on a PC don’t need to see anything else, especially not some incomprehensible JCL messages.

The pchost program inserts records into account entries table2 and updates records into account balance table1. If any record received from the PC is wrong for any reason (not numeric, wrong account number, wrong currency, etc), the program will not execute; it produces a return code, and returns the appropriate message to the PC. If something went wrong on the host side, an errors file will be written and so provide complete information to the developer, and return an appropriate ‘Program error’ message to the PC.

Here is its simplified code:

```
pchost: PROC OPTIONS(MAIN);
EXEC SQL INCLUDE table1;  /* account balance table
                                  definition and record description */
EXEC SQL INCLUDE table2;  /* account entries history table -"-       */
EXEC SQL INCLUDE SQLDA;
EXEC SQL INCLUDE SQLCA;
DCL filein FILE INPUT;
DCL errors FILE OUTPUT;
DCL errors_line CHAR(8Ø);
DCL 1 record_desc,
     2  account_number PIC'(9)9',
       2  currency              CHAR(3),
      2  amount               PIC'(5)Z9V.99';
DCL wrong_record CHAR(21)  BASED(ADDR(record_desc));
DCL end_of_file FIXED(1) INIT(Ø);
DCL (NULL,SUBSTR,VERIFY,ADDR,DATETIME,STG,CSTG,PLIRETC) BUILTIN;
```

```
DCL picsql PIC'-9';
/************************ PROGRAM *********************************/

ON ENDFILE(filein) end_of_file=1;
READ FILE(filein) INTO(record_desc);
DO WHILE (end_of_file =0);

  CALL check_record;
  table1.account_number=record_desc.account_number;
  table1.currency=  record_desc.currency;
  EXEC SQL SELECT account_balance
           INTO :table1.account_balance
           FROM table1
           WHERE account_number=:table1.account_number ;
  SELECT (SQLCODE);
     WHEN(0);
     WHEN(100)
     DO;
         EXEC SQL ROLLBACK WORK;

PUT SKIP EDIT('Unknown account number '!!record_desc.account_number)(A);
        CALL exit;
     END;
     OTHER DO;
        EXEC SQL ROLLBACK WORK;
        PUT SKIP EDIT('Program error !!')(A);
        picsql=SQLCODE;
        errors_line='SELECT table1 error. Account number
'!!record_desc.account_number!!' code = '
                !!picsql;
        WRITE FILE (errors) FROM (errors_line);
        CALL exit;
    END;
 END;
 account_balance =  account_balance + record_desc.amount;
  EXEC SQL UPDATE table1
              SET account_balance =: account_balance
              WHERE account_number =: account_number;
  IF SQLCODE^=0 THEN
  DO;
       EXEC SQL ROLLBACK WORK;
       PUT SKIP EDIT('Program error')(A);
       picsql=SQLCODE;
       errors_line='UPDATE table1 error. Account number
'!!record_desc.account_number!!' code = '
                !!picsql;
       WRITE FILE (errors) FROM (errors_line);
       CALL exit;
   END;
   CALL write_table2;
```

```
   READ FILE(filein) INTO(record_desc);
END; /* DO WHILE(EOF=Ø) */
PUT SKIP EDIT('DONE')(A);    /* successful end of program */
/******************** PROCEDURE ************************/
check_record: PROC;
     /* checks the account number, currency code and amount validity  */
    /* if anything is wrong, writes sysprint file end exit the program*/
    IF VERIFY(record_desc.account_number,'Ø123456789')^=Ø !
         VERIFY(record_desc.amount,'Ø123456789. ')^=Ø  ! wrong currency
THEN
    DO;
        EXEC SQL ROLLBACK WORK;
        PUT SKIP EDIT('wrong record '!! wrong record)(A);
        CALL exit;
    END;
END check_record;
write_table2: PROC;
  EXEC SQL INSERT INTO table2 VALUES
   ( :table1.account_number,    CURRENT_DATE,
     :CURRENT_TIME,           :record_desc.currency,
     :record_desc.amount,     :table1.account_balance
      …  );
   IF SQLCODE^=Ø THEN
   DO;
      EXEC SQL ROLLBACK WORK;
      PUT SKIP EDIT('Program error')(A);
      picsql=SQLCODE;
      errors_line='INSERT table2 error. Account number
'!!record_desc.account_number!!' code = '
                !!picsql;
      WRITE FILE (errors) FROM (errors_line);
      CALL exit;
   END;
END write_table2;
exit: PROC;  /* raises return code for a whole beta92 jcl report */
    CALL PLIRETC(Ø9);
    GOTO end;
END exit;
end: END pchost;
```

CONCLUSION

Whenever there is a need to transfer files from a PC to a host, process the transferred data, and to get the results back on the PC, it is possible to do it in the following manner:

- Prepare a file for transfer on the PC side, in the simplest way possible.

- Start a PC bat procedure that transfers the file using FTP, starts a job (program) on the host side, and returns its result in the most appropriate way to the PC user.
- Write both JCL and the program on the host side. Messages that the program produces should be separated for PC users and developers and written so they are useful for both.

Vladimir Antonijevic
IT Analyst
Postal Savings Bank (Yugoslavia)

SMTP mail techniques

As TCP/IP became more popular on the mainframe, we began to exploit SMTP services. This article provides several examples of how to use SMTP to enhance batch processing, as well as interactive use. It assumes that you have SMTP configured correctly on your MVS system. Please refer to the IBM *CS IP Configuration Reference* manual for SMTP configuration details.

Over the years I had many little projects to supplement existing mainframe processes with e-mail support. Some of these were simply to add a condition code driven IEBGENER step to an existing job that pointed to a SYSUT1 dataset containing the text of an e-mail and the SYSUT2 pointing to the SMTP external writer:

```
//         IF (ABEND | RC > Ø) THEN
//MAILIT   EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=*
//SYSUT1   DD   DSN=my.ctllib(email),DISP=SHR
//SUSYT2   DD   SYSOUT=(B,SMTP)
//SYSIN    DD   DUMMY
//         ENDIF
```

This technique was very static, each job had to have a

specifically tailored e-mail CTLLIB member. After a while I wrote some little REXX EXECs to be used in batch to generate the SMTP envelope and accept parameters for various purposes. All the batch JCL for the next three examples is the same:

```
//         IF (ABEND | RC > Ø) THEN
//MAILIT   EXEC PGM=IKJEFTØ1,PARM='progname progparms'
//SYSEXEC  DD   DSN=my.exec.pds,DISP=SHR
//SYSTSPRT DD   SYSOUT=*
//SYSTSIN  DD   DUMMY
//         ENDIF
```

The first program is used to send a ‘canned’ message to an e-mail address or a pager. The subject will be “<jobname> failed MM/DD/YY HH:MM” and the text will be “Call the Data Center immediately 888-333-4444”. Most pagers have an e-mail address associated with the phone number (ie 8881112222@pagerco.com). The first mini program is called PAGER. The only parameter is the e-mail address of the recipient.

```
/* REXX - PAGER */
 parse arg oncall
 mail.1 = 'HELLO' MVSVAR('SYSNAME')
 mail.2 = 'MAIL FROM:<yourid@anywhere.com>'
 mail.3 = 'RCPT TO:<'oncall'>'
 mail.4 = 'DATA'
 mail.5 = 'FROM:<yourid@anywhere.com>'
 mail.6 = 'Subject:' MVSVAR('SYMDEF','JOBNAME') 'failed' date() time()
 mail.7 = 'Call the Data Center immediately 888-333-4444'
 mail.8 = 'QUIT'
"ALLOC F(MAIL) SYSOUT(B) WRITER(SMTP)"
 if RC <> Ø then say 'ALLOC error on MAIL'
"EXECIO * DISKW MAIL (STEM MAIL. FINIS"
 if RC <> Ø then say 'EXECIO error on MAIL'
"FREE F(MAIL)"
```

The JCL parameter string from above would be:

```
PARM='PAGER 8881112222@pagerco.com'
```

The second program is used to send a small message to an e-mail address or pager. This could be used from TSO or from batch. This second mini program is called MSG and has two parameters – the e-mail address and the message text.

Remember, the limit for JCL PARMs is 100 characters. If this is run from TSO, REXX uses a 4-byte length for input arguments, so there is no real limit to what you could type into this EXEC from a command line.

```
/* REXX - MSG */
 parse arg id msg
 mail.1 = 'HELLO' MVSVAR('SYSNAME')
 mail.2 = 'MAIL FROM:<yourid@anywhere.com>'
 mail.3 = 'RCPT TO:<'id'>'
 mail.4 = 'DATA'
 mail.5 = 'FROM:<yourid@anywhere.com>'
 mail.6 = 'Subject: Call the Data Center: 888-333-4444'
 mail.7 =  msg
 mail.8 = 'QUIT'
"ALLOC F(MAIL) SYSOUT(B) WRITER(SMTP) REU"
 if RC <> 0 then say 'ALLOC error on MAIL'
"EXECIO * DISKW MAIL (STEM MAIL. FINIS"
 if RC <> 0 then say 'EXECIO error on MAIL'
"FREE F(MAIL)"
```

The JCL parameter string from above would be:

```
PARM='MSG 8881112222@pagerco.com Any text that does not exceed the parm
limit'
```

The third program is a combination of the first two. It is intended to be used in batch and allows the JCL coder to add both the on-call e-mail address and a short message in the JCL. A minor enhancement in this EXEC is the addition of the control block chasing to acquire the job number and the e-mail will always appear to come from the LPAR that the job ran on. This allows jobs to contain multiple notification steps that are tailored to the specific steps that failed (or succeeded, if a progress e-mail/ page is desired). This technique can be strategically combined with PROCs, symbolics, and INCLUDEs to minimize the changes to any JCL when the on-call person changes. We used a single member to hold the current on-call e-mail address and this member was included in all the production JCL supported by that on-call member. The on-call member was updated by the person ending their on-call session (a little incentive to keep it current!).

```
/* REXX - NOTIFY */
 parse arg oncall message
```

```
 if oncall = '' then exit(2Ø)
 if message = '' then message = 'No additional message'
 tcb     = storage(21c,4)
 jscb    = storage(d2x(c2d(tcb)+18Ø),4)
 ssib    = storage(d2x(c2d(jscb)+316),4)
 jobnum  = strip(storage(d2x(c2d(ssib)+15),5),l,Ø)
 subject =  mvsvar('SYMDEF','JOBNAME') 'J'jobnum 'failed' date() time()
 mail.1  = 'HELO' mvsvar('SYSNAME')
 mail.2  = 'MAIL FROM:<'mvsvar('SYSNAME')'>'
 mail.3  = 'RCPT TO:<'oncall'>'
 mail.4  = 'DATA'
 mail.5  = 'FROM:<'mvsvar('SYSNAME')'>'
 mail.6  = 'Subject:' subject
 mail.7  =  message
 mail.8  = 'QUIT'
"ALLOC F(MAIL) SYSOUT(B) WRITER(SMTP) REU"
 EXITRC = RC
 if EXITRC <> Ø then say 'ALLOC error on MAIL'
"EXECIO * DISKW MAIL (STEM MAIL. FINIS"
 EXITRC = RC
 if EXITRC <> Ø then say 'EXECIO error on MAIL'
"FREE F(MAIL)"
 exit(EXITRC)
```

The JCL parameter string from the above would be:

```
PARM='NOTIFY 8881112222@pagerco.com any old text that fits in the JCL
parm'
```

Sample PROC for NOTIFY:

```
//NOTIFY   PROC ID=,MSG=
//NOTIFY   EXEC PGM=IKJEFTØ1,PARM='NOTIFY &ID &MSG'
//SYSEXEC  DD   DSN=my.exec.pds,DISP=SHR
//SYSTSPRT DD   SYSOUT=*
//SYSTSIN  DD   DUMMY
```

The execution JCL contains the following code:

```
//         IF (ABEND | RC > Ø) THEN
//NOTIFY   EXEC NOTIFY,MSG='this is the message text',
//         INCLUDE MEMBER=ONCALL
//         ENDIF
```

The ONCALL INCLUDE member contains the following:

```
//         ID='8881112222@pagerco.com'
```

The same INCLUDE technique could be used for multiple MSG members to contain several reusable message text members.

Admittedly, it gets a little complicated after a while and difficult for new support people to follow.

Minor modifications could easily support an ONCALL PDS with a member for all on-call groups, and each member would simply contain the e-mail address or pager number of the unlucky person. This could become even more sophisticated with a date look-up so the on-call schedule could be built far into the future and also used as a reference. We ended up dismantling all job-based paging because our production control group was getting bypassed and things started slipping between the cracks. We did keep the e-mail notification technique for status and progress updates, but failures are handled first by production control and escalated to a support person if necessary. This pretty much ended development on my paging EXECs.

Even though I had learned all these great techniques, what I really wanted to do was mail an MVS dataset, a spool dataset, or HFS file directly or possibly send it as an attachment. This would be great for sharing or demonstrating something, or making a point. Originally this was done using another IEBGENER job or the following technique in the ISPF Editor:

1 Toggle to my emulator

2 Scroll to the area to copy

3 Highlight the area to copy

4 Copy the screen (Cntl-C)

5 Toggle to Word or e-mail

6 Paste the screen (Cntl-V)

7 Repeat for all the pages.

Then it hit me one day: write an ISPF edit macro that uses SMTP to do the job, and @MAIL was born. @MAIL is an ISPF edit macro that can be used in any ISPF edit or view session. It provides a good set of features that deal with most requirements:

1 Send the current edit session as the text of an e-mail.

2 Send the current edit session as a MIME encoded text attachment to an e-mail.

3 Send the current edit session as either text or an attachment to a distribution list.

The early versions of @MAIL did not support attachments. The MIME encoding looked a little tricky at first, but after reading the RFCs, it turned out to be pretty simple for TEXT and HTML. My @MAIL macro supports only TEXT attachments. Other formats require the attached data actually to be in a different format. The MIME subroutines used in @MAIL will support TEXT and HTML attachments, and I actually use them in other REXX EXECs to e-mail HTML-ized Infoman records.

To use @MAIL, simply type @MAIL on the edit or view command line.

```
   File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

 EDIT       OPSROZ.JCL(IEBGENER) - 01.04            Columns 00001 00072
  Command ===> @mail                                   Scroll ===> CSR
  ****** ************************ Top of Data ************************
 000001 //OPSROZJ  JOB
(0000),'IEBGENER',CLASS=A,MSGCLASS=T,NOTIFY=&SYSUID
 000002 /*JOBPARM SYSAFF=*
 000003//
*********************************************************************
```

After pressing *Enter*, a pop-up appears to collect the following:

1 Who to send the e-mail to (recipient).

2 Who it is coming from (sender – defaults to DEFAULT@your.domain.com).

3 The subject of the e-mail (defaults for DSNs and blank for HFS and SDSF).

4 Send this as an attachment (YES or NO).

5 If the recipient does not contain an @, assume it is a distribution list member and you must provide a valid PDS for the member.

```
File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help

EDIT       OPSROZ.JCL(IEBGENER) - Ø1.Ø4              Columns ØØØØ1 ØØØ72
Command ===> @mail                                     Scroll ===> CSR
****** ************************** Top of Data ************************
Ø +------------------ Enter @MAIL Parameters ------------------+ &SYSUID
Ø | Edit session contents will be sent by SMTP Email           |
Ø | Recipient:  robert.zenuk@anydomain.com                     |
************
Ø | Sender   :  DEFAULT@mydomain.com                           |  NER)
Ø | Subject  :  OPSROZ.JCL(IEBGENER)                           |  ------
Ø |                                                            |
Ø | Send as a text attachment:  NO  ( YES or NO )              |
************
Ø |                                                            |
Ø | To use distribution lists, set Recipient to a member       |
Ø | name without a @ and provide the PDS name that contains    |  R
Ø | the distribution member                                    |
,UNIT=SYSDA,
Ø | DSN      :                                                 |
Ø +------------------------------------------------------------+
ØØØØ14 //COPYØ96  EXEC PGM=IEBGENER
ØØØØ15 //SYSPRINT DD   SYSOUT=*
ØØØØ16 //SYSUT1   DD   DSN=SYSØ.SYSLOG.DAILY.SY36.ARCØ3Ø96,DISP=SHR
ØØØØ17 //SYSUT2   DD
DSN=OPSROZ.SY36.SYSLOG.JØ3Ø96,DISP=(,CATLG),UNIT=SYSDA,
ØØØØ18 //         SPACE=(CYL,(1ØØ,2Ø),RLSE)
ØØØØ19 //SYSIN    DD   DUMMY
****** ************************** Bottom of Data ***********************
```

The most recent recipient, sender, and attachment values are stored in your ISPF profile.

Aside from the normal use of @MAIL with edit and view, here are some additional very useful opportunities:

1 HFS files under USS IShell.

2 Submitted JCL under SDSF using the SDSF SJ line command.

3 Job output under SDSF using the SDSF SE command.

4 After using SDSF ULOG, use PRINT;PRINT CLOSE and the SDSF SE line command on the userid output in the Hold queue (great for operations documentation).

5 Any other ISPF application that provides edit or view capabilities.

Distribution lists are also easy to use. A PDS member serves as a distribution list. The format is fairly free format and simple:

```
To first.last@address.com
Cc another.name@another.com
Bcc yet.another@somwhere.com
```

Any number of entries for each type (to, cc, and bcc) is supported and distribution lists can be reused. Notice there are no colons following the to, cc, and bcc keywords. Case is not important and an asterisk in column 1 denotes a comment.

Many at our shop have found this useful. Hopefully, you will too.

```
/*********************************************************************/
/*                               REXX                                */
/*********************************************************************/
/* Purpose: Mail the contents of this edit session                   */
/*-------------------------------------------------------------------*/
/* Syntax: @MAIL smtpto smtpsub smtpfrom                             */
/*-------------------------------------------------------------------*/
/* Parms: smtpto     - The Email id of the recipient                 */
/*        smtpsub    - The Email Subject                             */
/*        smtpfrom   - The Email id of the sender                    */
/*********************************************************************/
 call time 'r'
 parse arg parms
 signal on syntax name trap
 signal on failure name trap
 signal on novalue name trap
 probe = 'NONE'
 modtrace = 'NO'
 modspace = ''
 call stdentry 'DIAGMSGS'
 module = 'MAINLINE'
 push trace() time('L') module 'From:' Ø 'Parms:' parms
 if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
 call modtrace 'START' Ø
 @vio   = 'VIO'
 @sysda = 'SYSDA'
 call isrwrap "MACRO (SMTPTO SMTPSUB SMTPFROM)"
 call isrwrap "(DSN) = DATASET"
 call isrwrap "(MEM) = MEMBER"
 if smtpto <> '' then call ispwrap "VPUT (SMTPTO) PROFILE"
 if smtpfrom <> '' then call ispwrap "VPUT (SMTPFROM) PROFILE"
 call ispwrap 8 "VGET (SMTPTO) PROFILE"
 call ispwrap 8 "VGET (SMTPFROM) PROFILE"
 call ispwrap 8 "VGET (SMTPATT) PROFILE"
 if smtpfrom = '' then smtpfrom = 'DEFAULT'
```

```
if mem <> '' then
   smtpsub = dsn'('mem')'
else
   smtpsub = dsn
smtp.1  = ")ATTR                                                    "
smtp.2  = " ~ TYPE(TEXT) COLOR(TURQ)                                "
smtp.3  = " # TYPE(INPUT) CAPS(OFF) COLOR(GREEN)                    "
smtp.4  = " $ TYPE(INPUT) CAPS(ON) COLOR(RED)                       "
smtp.5  = ")BODY EXPAND(//) WINDOW(60,11)                           "
smtp.6  = "%Edit session contents will be sent by SMTP Email        "
smtp.7  = "~Recipient: #Z                                           "
smtp.8  = "~Sender   : #Z                                           "
smtp.9  = "~Subject  : #Z                                           "
smtp.10 = "                                                         "
smtp.11 = "~Send as a text attachment: $Z  +(%YES+or%NO+)           "
smtp.12 = "                                                         "
smtp.13 = "~To use distribution lists, set Recipient to a member name"
smtp.14 = "~without a%@~and provide the PDS name that contains the  "
smtp.15 = "~distribution member                                     "
smtp.16 = "~DSN      : $Z                                           "
smtp.17 = ")INIT                                                    "
smtp.18 = " .ZVARS = '(SMTPTO SMTPFROM SMTPSUB SMTPATT SMTPDSN)'    "
smtp.19 = " IF (&SMTPFROM = &Z)                                     "
smtp.20 = "     &SMTPFROM = &ZUSER                                  "
smtp.21 = " IF (&SMTPATT  = &Z)                                     "
smtp.22 = "     &SMTPATT  = 'NO'                                    "
smtp.23 = ")PROC                                                    "
smtp.24 = " VER (&SMTPTO,NB)                                        "
smtp.25 = " VER (&SMTPFROM,NB)                                      "
smtp.26 = " VER (&SMTPSUB,NB)                                       "
smtp.27 = " VER (&SMTPATT,NB,LIST,YES,NO)                           "
smtp.28 = " IF  (VER (&SMTPTO,LEN,'<=',9))                          "
smtp.29 = "      VER (&SMTPDSN,NB,DSNAMEQ)                          "
smtp.30 = ")END                                                     "
call popdyn 'SMTP' 4 'Enter' execname 'Parameters'
call ispwrap "VPUT (SMTPTO) PROFILE"
call ispwrap "VPUT (SMTPFROM) PROFILE"
call ispwrap "VPUT (SMTPATT) PROFILE"
if pos('@',smtpto) = 0 then
   do
    distdsn = strip(smtpdsn,"B","'")||'('||smtpto||')'
    call tsotrap "ALLOC F("smtpto") DA("qdsn(distdsn)") SHR REUSE"
    call msg 'Enter email addresses on separate lines prefixing with',
             'TO or CC or BCC and at least one space'
    call ispwrap 4 "EDIT DATASET("qdsn(distdsn)")"
   end
if smtpatt = 'YES' then
   do
    m = mailhdr('SMTPDOC' smtpfrom smtpto 'HTML' smtpsub)
    m = stem('SMTPDOC' m 'Dataset was sent as an attachment')
```

```
    if words(smtpsub) > 1 then
       smtpfnam = word(smtpsub,1)
    else
       smtpfnam = smtpsub
    m = mimefile('SMTPDOC' m smtpfnam'.txt')
   end
else
   m = mailhdr('SMTPDOC' smtpfrom smtpto 'NO' smtpsub)
call isrwrap "CURSOR = .ZLAST"
call isrwrap "(LAST) = CURSOR"
do i=1 to last
   call isrwrap "(LINE) = LINE" i
   m = stem('$SMTPDOC' m strip(line,'T'))
end
call mailit 'SMTPDOC' m
call ispwrap "REMPOP"
if pos('@',smtpto) = Ø then
   if listdsi(smtpto "FILE") = Ø then
      call tsotrap "FREE F("smtpto")"
call msg 'Subject: "'smtpsub'" sent to:' smtpto', by: 'smtpfrom
shutdown: nop
          if isr_verb <> 'MACRO' & ERC <> 2Ø then
             do
              call outtrap 'toss.'
              address TSO "FREE F(EMAIL)"
              call outtrap 'off'
             end
          EXITRC = 1
          call stdexit time('e')
rcexit: parse arg EXITRC zedlmsg
        if EXITRC <> Ø then
           do
            trace 'o'
            if execenv = 'TSO' | execenv = 'ISPF' then
               do
                if ispfenv = 'YES' then
                   do
                    zispfrc = EXITRC
                    address ISPEXEC "VPUT (ZISPFRC)"
                   end
               end
            if zedlmsg <> '' then
               do
                zedlmsg = time('L') execname zedlmsg 'RC='EXITRC
                call msg zedlmsg
               end
            stacktitle = 'Parentage Stack Trace ('queued()' entries):'
            if tsoenv = 'BACK' then
               do
                if subword(zedlmsg,9,1) = msgdd then
```

```
                  do
                   say zedlmsg
                   signal shutdown
                  end
                else
                  do
                   call saydd msgdd 1 zedlmsg
                   call saydd msgdd 1 stacktitle
                  end
               end
            else
               do
                zerrlm = zedlmsg
                address ISPEXEC "LOG MSG(ISRZØØ3)"
                zerrlm = center(' 'stacktitle' ',78,'-')
                address ISPEXEC "LOG MSG(ISRZØØ3)"
               end
            do queued()
               pull stackinfo
               if tsoenv = 'BACK' then
                  do
                   call saydd msgdd Ø stackinfo
                  end
               else
                  do
                   zerrlm = stackinfo
                   address ISPEXEC "LOG MSG(ISRZØØ3)"
                  end
            end
            if tsoenv = 'FORE' then
               do
                zerrlm = center(' 'stacktitle' ',78,'-')
                address ISPEXEC "LOG MSG(ISRZØØ3)"
               end
            signal shutdown
           end
        else
           return
trap: traptype = condition('C')
      if traptype = 'SYNTAX' then
         msg = errortext(RC)
      else
         msg = condition('D')
      trapline = strip(sourceline(sigl))
      msg = traptype 'TRAP:' msg', Line:' sigl '"'trapline'"'
      call rcexit 666 msg
errmsg: nop
        parse arg errline text
        return 'Error on statement' errline',' text
stdentry: module = 'STDENTRY'
```

```
if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
parse arg sparms
push trace() time('L') module 'From:' sigl 'Parms:' sparms
arg msgdd
parse upper source . . execname . execdsn . . execenv .
EXITRC = Ø
MAXRC = Ø
ispfenv = 'NO'
popup = 'NO'
lockpop = 'NO'
headoff = 'NO'
hcreator = 'NO'
keepstack = 'NO'
lpar = mvsvar('SYSNAME')
zedlmsg = 'Default shutdown message'
if substr(execenv,1,3) <> 'TSO' & execenv <> 'ISPF' then
   tsoenv = 'NONE'
else
   do
    tsoenv = sysvar('SYSENV')
    signal off failure
   "ISPQRY"
    ISPRC = RC
    if ISPRC = Ø then
       do
        ispfenv = 'YES'
        call ispwrap "VGET (ZSCREEN)"
        if tsoenv = 'BACK' then
           htable = jobinfo(1)||jobinfo(2)
        else
           htable = userid()||zscreen
        TBCRC = ispwrap(8 "TBCREATE" htable "KEYS(HEAD)")
        if TBCRC = Ø then
           do
            headoff = 'NO'
            hcreator = 'YES'
           end
        else
           do
            headoff = 'YES'
           end
       end
    signal on failure name trap
   end
call modtrace 'START' sigl
startmsg = execname 'started' date() time() 'on' lpar
if tsoenv = 'BACK' & sysvar('SYSNEST') = 'NO' &,
   headoff = 'NO' then
   do
    jobname = mvsvar('SYMDEF','JOBNAME')
```

```
     jobinfo = jobinfo()
     parse var jobinfo jobtype jobnum .
      say jobname center(' 'startmsg' ',61,'-') jobtype jobnum
     say
     if ISPRC = -3 then
        do
         call saydd msgdd 1 'ISPF ISPQRY module not found,',
                            'ISPQRY is usually in the LINKLST'
         call rcexit 2Ø 'ISPF ISPQRY module is missing'
        end
     if msgdd <> '' then
        do
         call ddcheck msgdd
         call saydd msgdd 1 startmsg
         call ddcheck 'SYSEXEC'
         call saydd msgdd Ø execname 'loaded from' sysdsname
         if parms <> '' then
            call saydd msgdd Ø 'Parms:' parms
         if listdsi('STEPLIB' 'FILE') = Ø then
            do
             steplibs = dddsns('STEPLIB')
             call saydd msgdd Ø 'STEPLIB executables loaded',
                 'from' word(dddsns,1)
             if dddsns('STEPLIB') > 1 then
                do
                 do stl=2 to steplibs
                    call saydd msgdd Ø copies(' ',31),
                         word(dddsns,stl)
                 end
                end
            end
        end
   end
else
   do
    fkaset = 'OFF'
    call ispwrap "VGET (ZFKA) PROFILE"
    if zfka <> 'OFF' then
       do
        fkaset = zfka
        fkacmd = 'FKA OFF'
        call ispwrap "CONTROL DISPLAY SAVE"
       call ispwrap "DISPLAY PANEL(ISPBLANK) COMMAND(FKACMD)"
        call ispwrap "CONTROL DISPLAY RESTORE"
       end
   end
pull tracelvl . module . sigl . sparms
call modtrace 'STOP' sigl
interpret 'trace' tracelvl
return
```

```
stdexit: module = 'STDEXIT'
         if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
         arg endtime
         endmsg = execname 'ended' date() time() format(endtime,,1)
         if MAXRC > EXITRC then EXITRC = MAXRC
         endmsg = endmsg 'on' lpar 'RC='EXITRC
         if tsoenv = 'BACK' & sysvar('SYSNEST') = 'NO' &,
            headoff = 'NO' then
            do
             say
             say jobname center(' 'endmsg' ',61,'-') jobtype jobnum
             if msgdd <> '' & subword(zedlmsg,9,1) <> msgdd then
                do
               call saydd msgdd 1 execname 'ran in' endtime 'seconds'
                 call saydd msgdd Ø endmsg
                end
            end
         else
            do
             if fkaset <> 'OFF' then
                do
                 fkafix = 'FKA'
                 call ispwrap "CONTROL DISPLAY SAVE"
               call ispwrap "DISPLAY PANEL(ISPBLANK) COMMAND(FKAFIX)"
                 if fkaset = 'SHORT' then
                    call ispwrap "DISPLAY PANEL(ISPBLANK)",
                                 "COMMAND(FKAFIX)"
                 call ispwrap "CONTROL DISPLAY RESTORE"
                end
            end
         if ispfenv = 'YES' & hcreator = 'YES' then
            call ispwrap "TBEND" htable
         call modtrace 'STOP' sigl
         if queued() > Ø then pull . . module . sigl . sparms
         if queued() > Ø then pull . . module . sigl . sparms
         if tsoenv = 'FORE' & queued() > Ø & keepstack = 'NO' then
            pull . . module . sigl . sparms
         if queued() > Ø & keepstack = 'NO' then
            do
             say queued() 'Leftover Parentage Stack Entries:'
             say
             do queued()
                pull stackundo
                say stackundo
             end
             EXITRC = 1
            end
```

```
         exit(EXITRC)
msg: module = 'MSG'
     parse arg zedlmsg
     if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
     parse arg sparms
     push trace() time('L') module 'From:' sigl 'Parms:' sparms
     call modtrace 'START' sigl
     if tsoenv = 'BACK' | execenv = 'OMVS' then
        say zedlmsg
     else
        do
         if ispfenv = 'YES' then
            address ISPEXEC "SETMSG MSG(ISRZØØØ)"
         else
            say zedlmsg
        end
     pull tracelvl . module . sigl . sparms
     call modtrace 'STOP' sigl
     interpret 'trace' tracelvl
     return
ddcheck: module = 'DDCHECK'
         if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
         arg dd
         dderrmsg = 'OK'
         LRC = listdsi(dd "FILE")
         if LRC <> Ø & strip(sysreason,'L',Ø) <> 3 then
            do
             dderrmsg = errmsg(sigl 'Required DD' dd 'is missing')
             call rcexit LRC dderrmsg sysmsglvl2
            end
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return
qdsn: module = 'QDSN'
      if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
      parse arg sparms
      push trace() time('L') module 'From:' sigl 'Parms:' sparms
      call modtrace 'START' sigl
      parse arg qdsn
      qdsn = "'"strip(qdsn,"B","'")"'"
      pull tracelvl . module . sigl . sparms
      call modtrace 'STOP' sigl
      interpret 'trace' tracelvl
      return qdsn
tempmem: module = 'TEMPMEM'
         if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
```

```
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg tempmem
          if length(tempmem) < 7 then
             tempdd = '$'tempmem'$'
          else
             tempdd = '$'substr(tempmem,2,6)'$'
          if listdsi(tempdd 'FILE') = 0 then "FREE F("tempdd")"
          tempdsn = pandsn(tempmem)
          parse var tempdsn temppds '(' .
          call tsotrap "ALLOC F("tempdd") DA("qdsn(tempdsn)") NEW",
                       "LRECL(80) BLKS(0) DIR(1) SPACE(1) CATALOG",
                       "UNIT("@sysda") RECFM(F B)"
          call tsotrap 'EXECIO * DISKW' tempdd '(STEM' tempmem'. FINIS'
          interpret 'drop' tempmem'.'
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return tempdd
ispwrap:  module = 'ISPWRAP'
          if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          parse arg ispparm
          zerrlm = 'NO ZERRLM'
          parse var ispparm valid_rc isp_cmd
          if datatype(valid_rc,'W') = 0 then
             do
              valid_rc = 0
              isp_cmd = ispparm
             end
          address ISPEXEC isp_cmd
          IRC = RC
          if IRC <= valid_rc then
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return IRC
             end
          else
             do
              perrmsg = errmsg(sigl 'ISPF Command:')
              call rcexit IRC perrmsg isp_cmd strip(zerrlm)
             end
isrwrap:  module = 'ISRWRAP'
          if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
          parse arg sparms
```

```
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          parse arg isrparm
          parse var isrparm valid_rc isr_cmd
          if datatype(valid_rc,'W') = Ø then
             do
              valid_rc = Ø
              isr_cmd = isrparm
             end
          parse var isr_cmd isr_verb .
          address ISREDIT isr_cmd
          ERC = RC
          if ERC <= valid_rc then
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return ERC
             end
          else
             do
              if isr_verb = 'MACRO' & ERC = 2Ø then
                 do
                  call rcexit ERC 'is an Edit Macro and not valid to',
                              'run outside of ISPF Edit'
                 end
              else
                 do
                  rerrmsg = errmsg(sigl 'ISPF Edit Command:')
                  call rcexit ERC rerrmsg isr_cmd
                 end
             end
tsotrap:  module = 'TSOTRAP'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          parse arg tsoparm
          parse var tsoparm valid_rc tso_cmd
          if datatype(valid_rc,'W') = Ø then
             do
              valid_rc = Ø
              tso_cmd = tsoparm
             end
          call outtrap 'tsoout.'
          tsoline = sigl
          address TSO tso_cmd
          CRC = RC
          call outtrap 'off'
          if CRC <= valid_rc then
```

```
            do
             pull tracelvl . module . sigl . sparms
             call modtrace 'STOP' sigl
             interpret 'trace' tracelvl
             return CRC
            end
          else
            do
             trapmsg = center(' TSO Command Error Trap ',78,'-')
             terrmsg = errmsg(sigl 'TSO Command:')
             if tsoenv = 'BACK' | execenv = 'OMVS' then
                do
                 say trapmsg
                 do c=1 to tsoout.Ø
                    say tsoout.c
                 end
                 say trapmsg
                 call rcexit CRC terrmsg tso_cmd
                end
             else
                do
                 if ispfenv = 'YES' then
                    do
                     zedlmsg = trapmsg
                     address ISPEXEC "LOG MSG(ISRZØØØ)"
                     do c=1 to tsoout.Ø
                        zedlmsg = tsoout.c
                        address ISPEXEC "LOG MSG(ISRZØØØ)"
                     end
                     zedlmsg = trapmsg
                     address ISPEXEC "LOG MSG(ISRZØØØ)"
                     call rcexit CRC terrmsg tso_cmd,
                          ' see the ISPF Log (Option 7.5) for details'
                    end
                 else
                    do
                     say trapmsg
                     do c=1 to tsoout.Ø
                        say tsoout.c
                     end
                     say trapmsg
                     call rcexit CRC terrmsg tso_cmd
                    end
                end
            end
wait: module = 'WAIT'
      if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
      parse arg sparms
      push trace() time('L') module 'From:' sigl 'Parms:' sparms
      call modtrace 'START' sigl
```

```
      arg seconds wmode
      if datatype(seconds,'W') = Ø then seconds = 1Ø
      RC = syscalls('ON')
      if tsoenv = 'FORE' & ispfenv = 'YES' then
         call lock seconds 'second wait was requested'
      if tsoenv = 'BACK' & wmode = '' then
         call saydd msgdd Ø seconds 'second wait was requested'
      address SYSCALL "SLEEP" seconds
      if tsoenv = 'FORE' & ispfenv = 'YES' then
        call unlock
      RC = syscalls('OFF')
      pull tracelvl . module . sigl . sparms
      call modtrace 'STOP' sigl
      interpret 'trace' tracelvl
      return
setbord: module = 'SETBORD'
         if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
         arg color
         if color = '' then color = 'YELLOW'
         ispopt11.1=")BODY                        "
         ispopt11.2="%Command ===>_ZCMD        + "
         ispopt11.3=")INIT                        "
         ispopt11.4="&ZCMD = ' '                  "
         ispopt11.5="VGET (COLOR) SHARED          "
         ispopt11.6="&ZCOLOR = &COLOR             "
         ispopt11.7=".RESP = END                  "
         ispopt11.8=")END                         "
         setdd = tempmem('ISPOPT11')
         call ispwrap "LIBDEF ISPPLIB LIBRARY ID("setdd") STACK"
         call ispwrap "VPUT (COLOR) SHARED"
         call ispwrap "SELECT PGM(ISPOPT) PARM(ISPOPT11)"
         call ispwrap "LIBDEF ISPPLIB"
         call tsotrap "FREE F("setdd") DELETE"
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return
lock: module = 'LOCK'
      if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
      parse arg sparms
      push trace() time('L') module 'From:' sigl 'Parms:' sparms
      call modtrace 'START' sigl
      parse arg lockmsg
      if lockmsg = '' then lockmsg = 'Please be patient'
      if tsoenv = 'FORE' then
         do
          if length(lockmsg) < 76 then
```

```
            locklen = length(lockmsg) + 2
          else
            locklen = 77
          if locklen <= 1Ø then locklen = 1Ø
          lock.1 = ")BODY EXPAND(//) WINDOW("locklen",1)"
          lock.2 = "%&LOCKMSG                    "
          lock.3 = ")END                         "
          call ispwrap "CONTROL DISPLAY LOCK"
          call popdyn 'LOCK' 8 execname 'Please be patient'
          lockpop = 'YES'
         end
       pull tracelvl . module . sigl . sparms
       call modtrace 'STOP' sigl
       interpret 'trace' tracelvl
       return
unlock: module = 'UNLOCK'
        if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
        parse arg sparms
        push trace() time('L') module 'From:' sigl 'Parms:' sparms
        call modtrace 'START' sigl
        if tsoenv = 'FORE' then
            do
             if lockpop = 'YES' then
                do
                 call ispwrap "REMPOP"
                 lockpop = 'NO'
                end
             if popup = 'YES' then
                do
                 call setbord 'BLUE'
                 call ispwrap "REMPOP"
                 popup = 'NO'
                end
            end
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return
pandsn: module = 'PANDSN'
        if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
        parse arg sparms
        push trace() time('L') module 'From:' sigl 'Parms:' sparms
        call modtrace 'START' sigl
        arg panel
        unique = right(time('s'),7,Ø)||'('||panel||')'
        pandsn = userid()||'.'||execname||'.'||panel||'.T'||unique
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return pandsn
```

```
popdyn: module = 'POPDYN'
        if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
        parse arg sparms
        push trace() time('L') module 'From:' sigl 'Parms:' sparms
        call modtrace 'START' sigl
        parse arg dyn dynrow dynmsg
        if dynrow = '' then dynrow = 1
        if dynmsg = '' then dynmsg = execname
        dynreturn = 'NO'
        if word(dynmsg,1) = 'RETURN' then
           parse var dynmsg dynreturn dynmsg
        dyndd = tempmem(dyn)
        call ispwrap "LIBDEF ISPPLIB LIBRARY ID("dyndd") STACK"
        call setbord 'YELLOW'
        if dyn = 'LOCK' & popup = 'NO' then
           popup = 'NO'
        else
           popup = 'YES'
        zwinttl = dynmsg
        call ispwrap "ADDPOP ROW("dynrow")"
        DRC = ispwrap(8 "DISPLAY PANEL("dyn")")
        call ispwrap "LIBDEF ISPPLIB"
        call tsotrap "FREE F("dyndd") DELETE"
        call setbord 'BLUE'
        if dynreturn = 'NO' then
           call rcexit DRC 'terminated by user'
        if dynreturn = 'RETURN' & DRC = 8 then
           do
            call ispwrap "REMPOP"
            popup = 'NO'
           end
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return DRC
saydd: module = 'SAYDD'
       if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
       parse arg sparms
       push trace() time('L') module 'From:' sigl 'Parms:' sparms
       call modtrace 'START' sigl
       parse arg msgdd msglines message
       if words(msgdd msglines message) < 3 then
          call rcexit 33 'Missing MSGDD or MSGLINES'
       if datatype(msglines) <> 'NUM' then
          call rcexit 34 'MSGLINES must be numeric'
       if tsoenv <> 'BACK' then
          do
           pull tracelvl . module . sigl . sparms
           call modtrace 'STOP' sigl
           interpret 'trace' tracelvl
```

```
            return
          end
        call ddcheck msgdd
        msgb = 1
        if msglines > Ø then
          do msgb=1 to msglines
            msgline.msgb = ' '
          end
        msgm = msgb
        if length(message) > 6Ø then
          do
           messst = lastpos(' ',message,6Ø)
           messseg = substr(message,1,messst)
           msgline.msgm = date() time() strip(messseg)
           message = strip(delstr(message,1,messst))
           do while length(message) > Ø
              msgm = msgm + 1
              if length(message) > 55 then
                 messst = lastpos(' ',message,55)
              if messst > Ø then
                 messseg = substr(message,1,messst)
              else
                 messseg = substr(message,1,length(message))
              msgline.msgm = date() time() 'CONT:' strip(messseg)
              message = strip(delstr(message,1,length(messseg)))
           end
          end
        else
          do
           if message = '@BLANK@' then
              msgline.msgm = ' '
           else
              msgline.msgm = date() time() strip(message)
          end
        if msglines > Ø then
          do msgt=1 to msglines
            msge = msgt + msgm
            msgline.msge = ' '
          end
        call tsotrap "EXECIO * DISKW" msgdd "(STEM MSGLINE. FINIS"
        drop msgline. msgb msgt msge
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return
jobinfo: module = 'JOBINFO'
         if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
```

```
         arg item
         tcb      = ptr(54Ø)
         ascb     = ptr(548)
         tiot     = ptr(tcb+12)
         jscb     = ptr(tcb+18Ø)
         ssib     = ptr(jscb+316)
         asid     = c2d(stg(ascb+36,2))
         jobtype  = stg(ssib+12,3)
         jobnum   = strip(stg(ssib+15,5),'L',Ø)
         stepname = stg(tiot+8,8)
         procstep = stg(tiot+16,8)
         program  = stg(jscb+36Ø,8)
         jobdata  = jobtype jobnum stepname procstep program asid
         if item <> '' & (datatype(item,'W') = 1) then
            do
             pull tracelvl . module . sigl . sparms
             call modtrace 'STOP' sigl
             interpret 'trace' tracelvl
             return word(jobdata,item)
            end
         else
            do
             pull tracelvl . module . sigl . sparms
             call modtrace 'STOP' sigl
             interpret 'trace' tracelvl
             return jobdata
            end
ptr: return c2d(storage(d2x(arg(1)),4))
stg: return storage(d2x(arg(1)),arg(2))
stem: module = 'STEM'
      if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
      parse arg sparms
      push trace() time('L') module 'From:' sigl 'Parms:' sparms
      call modtrace 'START' sigl
      parse arg stemvar stemnum stemval
      if stemvar = '' then call rcexit 91 'STEMVAR is missing'
      if stemnum = '' then call rcexit 92 'STEMNUM is missing'
      if datatype(stemnum) <> 'NUM' then
         call rcexit 94 'STEMNUM is not numeric "'stemnum'"'
      if substr(stemvar,1,1) = '$' then
         do
          stemvar = strip(stemvar,'L','$')
          if stemval = '' then stemval = ' '
         end
      else
         if stemval = '' then call rcexit 93 'STEMVAL is missing'
      start = pos('"',stemval)
      if start <> Ø then
         do forever
            stemval = insert('"',stemval,start)
```

```
            start = pos('"',stemval,start+2)
            if start = 0 then leave
          end
        interpret stemvar'.'stemnum '= "'stemval'"'
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return stemnum + 1
mailit: module = 'MAILIT'
         if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
         arg mailstem mailseq maildbg
        if mailstem = '' then call rcexit 35 'MAILIT MAILSTEM is missing'
        if mailseq  = '' then call rcexit 36 'MAILIT MAILSEQ is missing'
         if smtpfile <> 'NO' then mailseq = mimeterm(mailstem mailseq)
         mailseq = stem(mailstem mailseq '.')
         mailseq = stem('$'mailstem mailseq ' ')
         mailseq = stem(mailstem mailseq 'QUIT')
         if maildbg <> '' then
            do mdbg=1 to mailseq-1
               interpret 'say strip('mailstem'.mdbg,"T")'
            end
         call tsotrap "ALLOC F($EMAIL$) SYSOUT(B) WRITER(SMTP)"
         call tsotrap "EXECIO * DISKW $EMAIL$ (STEM" mailstem". FINIS"
         call tsotrap "FREE F($EMAIL$)"
         interpret 'drop' mailstem'.'
         call saydd msgdd 1 'MAILIT sent document in' mailstem
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return mailseq-1
mailhdr: module = 'MAILHDR'
          if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          parse arg smtpdoc smtpfrom smtpto smtpfile smtpsub
          if words(sparms) < 5 then call rcexit 'MAILHDR parms missing'
          if smtpfrom = 'DEFAULT' then
             do
              call sockinit
              smtpenv = sockget('GetHostname')'.'sockget('GetDomain')
              smtpfrom = userid()'.'jobinfo(1)||jobinfo(2)'@'smtpenv
              call sockterm
             end
          mls = stem(smtpdoc 1 'HELO' sockget('GetHostname'))
          mls = stem(smtpdoc mls 'MAIL FROM:<'smtpfrom'>')
          if pos('@',smtpto) = 0 then
```

```
   do
    if length(smtpto) > 8 | pos('.',smtpto) <> Ø then
     call rcexit 2Ø 'Invalid DD name;' smtpto 'for maillist'
    call ddcheck smtpto
   call tsotrap "EXECIO * DISKR" smtpto "(STEM SENDTO. FINIS"
    say smtpsub 'will be emailed to:'
    say
    mla = 1
    do mli=1 to sendto.Ø
       if substr(sendto.mli,1,1) <> '*' then
          do
           parse var sendto.mli mailhow mailrcpt .
           mailhow.mla  = mailhow
           mailrcpt.mla = mailrcpt
           mls = stem(smtpdoc mls 'RCPT TO:<'mailrcpt.mla'>')
           if translate(mailhow) <> 'TO'  &,
              translate(mailhow) <> 'CC'  &,
              translate(mailhow) <> 'BCC' then
              call rcexit 66 'Bad email entry in SENDTO',
                             sendto.mli '('mailhow')'
           else
              do
               say strip(substr(sendto.mli,1,72))
               mla = mla + 1
              end
          end
       else
          iterate
    end
   end
else
   do
    mls = stem(smtpdoc mls 'RCPT TO:<'smtpto'>')
   end
mls = stem(smtpdoc mls 'DATA')
mls = stem(smtpdoc mls 'From: <'smtpfrom'>')
if pos('@',smtpto) = Ø then
    do mlx=1 to mla-1
    mls = stem(smtpdoc mls mailhow.mlx': <'mailrcpt.mlx'>')
    end
else
   mls = stem(smtpdoc mls 'To: <'smtpto'>')
mls = stem(smtpdoc mls 'Subject:' smtpsub)
if smtpfile <> 'NO' then
   mls = mimeinit(smtpdoc mls)
else
   mls = stem('$'smtpdoc mls ' ')
pull tracelvl . module . sigl . sparms
call modtrace 'STOP' sigl
interpret 'trace' tracelvl
```

```
          return mls
mimeinit: module = 'MIMEINIT'
            if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
            parse arg sparms
            push trace() time('L') module 'From:' sigl 'Parms:' sparms
            call modtrace 'START' sigl
            parse arg smtpdoc misn mimetype mimechar
            if mimetype = '' then mimetype = 'plain'
            if mimechar = '' then mimechar = 'iso-8859-1'
            mimebdy = 'Mime.'execname'.'date('B')time('S')'.'lpar
            misn = stem(smtpdoc misn 'Mime-Version: 1.Ø')
          mime1 = 'Content-type: multipart/mixed; boundary="'mimebdy'"'
            misn = stem(smtpdoc misn mime1)
            misn = stem('$'smtpdoc misn ' ')
            mime2 = 'This is a multi-part message in MIME format.'
            misn = stem(smtpdoc misn mime2)
            misn = stem(smtpdoc misn '--'mimebdy)
          mime3 = 'Content-Type: text/'mimetype'; charset="'mimechar'"'
            misn = stem(smtpdoc misn mime3)
            mime4 = 'Content-Transfer-Encoding: 7bit'
            misn = stem(smtpdoc misn mime4)
            misn = stem('$'smtpdoc misn ' ')
            pull tracelvl . module . sigl . sparms
            call modtrace 'STOP' sigl
            interpret 'trace' tracelvl
            return misn
mimefile: module = 'MIMEFILE'
            if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
            parse arg sparms
            push trace() time('L') module 'From:' sigl 'Parms:' sparms
            call modtrace 'START' sigl
            parse arg smtpdoc misn mimefile mimetype mimechar
            if smtpdoc = '' then call rcexit 38 'SMTPDOC is missing'
         if misn = '' then call rcexit 39 'SMTPDOC SEQ number is missing'
            if mimefile = '' then mimefile = 'attachment.html'
            if mimetype = '' then mimetype = 'html'
            if mimechar = '' then mimechar = 'iso-8859-1'
            misn = stem(smtpdoc misn '--'mimebdy)
         mime1 = 'Content-Type: text/'mimetype'; charset="'mimechar'";'
            misn = stem(smtpdoc misn mime1)
           mime2 = '              name="'mimefile'"'
            misn = stem(smtpdoc misn mime2)
            mime3 = 'Content-Transfer-Encoding: 7bit'
            misn = stem(smtpdoc misn mime3)
            mime4 = 'Content-Disposition: attachment;'
            misn = stem(smtpdoc misn mime4)
            mime5 = '   filename="'mimefile'"'
            misn = stem(smtpdoc misn mime5)
            mime6 = 'Content-Description: "'mimefile'"'
            misn = stem(smtpdoc misn mime6)
```

```
          misn = stem('$'smtpdoc misn ' ')
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return misn
mimeterm: module = 'MIMETERM'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          parse arg smtpdoc misn
          misn = stem(smtpdoc misn '--'mimebdy'--')
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return misn
sockinit: module = 'SOCKINIT'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg sockname
          if sockname = '' then sockname = execname
          call socket 'Initialize', sockname
          parse var result EXITRC socketname .
          call rcexit EXITRC 'Socket initialization error for',
               socketname
          call saydd msgdd Ø 'Socket' socketname 'Initialized'
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return
sockget: module = 'SOCKGET'
         if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
         arg sockcall
         if sockcall = '' then call rcexit 99 'Missing Socket Get'
         sockval = word(Socket(sockcall),2)
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return sockval
sockterm: module = 'SOCKTERM'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg sockname
```

```
            if sockname = '' then sockname = execname
            call socket 'Terminate'
            parse var result EXITRC socketname .
            call rcexit EXITRC 'Socket Termination error for',
                 socketname
            call saydd msgdd Ø 'Socket' socketname 'Terminated'
            pull tracelvl . module . sigl . sparms
            call modtrace 'STOP' sigl
            interpret 'trace' tracelvl
            return
modtrace:   if modtrace = 'NO' then return
            arg tracetyp sigline
            tracetyp = left(tracetyp,5)
            sigline = left(sigline,5)
            if tracetyp = 'START' then
               modspace = substr(modspace,1,length(modspace)+1)
          traceline = modspace time('L') tracetyp module sigline sparms
            if tracetyp = 'STOP' then
               modspace = substr(modspace,1,length(modspace)-1)
            if ispfenv = 'YES' & tsoenv = 'FORE' then
               do
                zedlmsg = traceline
                address ISPEXEC "LOG MSG(ISRZØØØ)"
               end
            else
               say traceline
            return
```

Editor's note: a full version of the code, including comments, is available from our Web site at www.xephon.com/extras/mail.txt.

Robert Zenuk
Systems Programmer (USA)

FTPing tapes

Say what? Electronically transferring tapes? The concept is simple: how to eliminate the physical requirement of moving a tape, be it between LPARs or to some distant point on the planet.

Since writing 'Moving datasets between systems' (see *TCP/*

SNA Update, issue 54, June 2004) I have had a chance to move a 39.5GB VTS (IBM's Virtual Tape Server) dataset, physically residing on a couple of tapes, to a single higher density non-VTS tape, still within the same ATL (Automatic Tape Library), but accessible from another LPAR.

FTP CONFIGURATION ISSUES

Allowing automatic tape mounts by FTP is a bit confusing. AUTOMOUNT is the SITE command required to allow the remote site to mount tapes, as required. But AUTOTAPEMOUNT TRUE is the remote system's FTP server configuration file (FTP.DATA) command required to allow tape mounts to actually occur. There is an FTP.DATA command AUTOMOUNT TRUE, but it indicates whether DASD volumes can be mounted!

Arguably, on a production system, you could endanger batch processing by stealing tape drives from it. This makes a fairly strong argument against allowing on-line tape mounts on production LPARs.

THE JCL

To transfer a tape between production and test LPARs, assuming that on-line tape mounts are allowed only on the test LPAR, log on to the production system and FTP to or from the test system. The following JCL copies a catalogued dataset on tape from production to test, giving it the same dataset name:

```
//FTPTAPE JOB TIME=10
//FTP     EXEC PGM=FTP,PARM='10.1.4.19',TIME=10
//NETRC    DD  DSN=E667800.ACTIVE.CNTL(FTPNETRC),DISP=SHR
//OUTPUT   DD  SYSOUT=*
//INPUT    DD  *
ebcdic
block
site autom unit=3590E vol=
put //DD:DODS1    'E972497.GROUP00.OSMOTS32.BKUP.D040406'
quit
//DODS1    DD  DSN=E972497.GROUP00.OSMOTS32.BKUP.D040406,DISP=SHR
//
```

The TIME=10 parameters on the JOB and EXEC statements were required to override the installation's default of one CPU minute for CLASS=A. Copying 39.5GB required 2.3 minutes of CPU time and one hour of elapsed time. SITE UNIT=3590E allowed it to fit on a single 40GB 3590E MagStar cartridge.

PUT //DD:DODS1 and the corresponding DD statement overcomes the AUTOMOUNT restriction on the production system. SITE AUTOM allows mounts on the test system.

The null VOL= specification overcomes a known problem with FTP that defaults VOL= to a value that causes the destination FTP server to fail.

VTS

Either Native or VTS tape (SITE UNIT=VTAPE) can be used. But there are restrictions on how many tape volumes can be created for a single dataset. For example, five is this installation's limit for VTS.

If output to VTS had been preferred, using this site's SMS classes, the SITE command would be changed to read:

```
site autom mgmtclass=MCTAPE stor=SCTAPE dataclass=DCVTS VOL=
```

Finally, if you are not familiar with the NETRC DD statement in FTP, it can be used as a single source of ID and password information, separate from the FTP commands. It also avoids, as in this case, including your password (BILLY in this example) in the JCL:

```
MACHINE 10.1.4.19 LOGIN E667800 PASSWORD BILLY
```

HOW NOT TO DO IT

Is eliminating physical installation tapes a good idea? Only if the FTP process is done properly. One major software vendor has made it so difficult that you will be tempted to quit half way through, pick up the phone, call your vendor rep and say, "Just FedEx the stupid tape".

I recently had to:

- Get signed up for the third incarnation of their support Web site in a year.
- Call their Help Desk because the log-on server would not respond for several hours, only to be told that it wouldn't work until I emptied my browser's Internet cache.
- Build a profile indicating what software products I had installed.
- Find the update section so I could select from the upgrades I was entitled to.
- Discover that I could download only a software release with the latest Service Pack installed, even though the release with Service Pack had not yet been certified on my release (1.4) of z/OS.
- Spend a long time downloading the 150MB zip file to my Windows 2000 workstation.
- Unzip it, only to discover that over half of it was not the product at all, but the latest version of the vendor's common platform for all their products, even though I do not plan to upgrade my current version of the common platform.
- Locate some documentation on the support site that tells me how to preallocate the mainframe dataset for the product file.
- FTP the product file to the mainframe.
- Locate some documentation on the support site that tells me what to do with the mainframe dataset.
- FTP the program to translate the mainframe dataset from the vendor's support site directly to the mainframe.
- Link edit the translation program into a Load Library.
- Run the translation program.
- Assign a tape drive so the program can mount a tape.
- Watch the program fail.

- Discover that I need BLP (Bypass Label Processing) access to write the tape.
- Figure out how to use RACF from a SPECIAL ID to give my normal ID BLP capabilities.
- Re-run the translation program.

All to end up with an installation tape for a single product where the physical VOLSER does not match the magnetic VOLSER on the tape.

Jon E Pearkins
Adiant (Canada)

HiperSockets – the pinnacle of mainframe TCP/IP

HiperSockets, introduced with the groundbreaking 64-bit z900 mainframe in October 2000 (though it took nearly a year before it was first made available with z/OS Version 1.2), is the ultimate proof, if one is now needed, of TCP/IP's role and significance *vis-à-vis* the future of mainframe computing. HiperSockets is a high-speed, near zero-latency, TCP/IP-specific, 'network-inside-a-mainframe'. It is implemented entirely by means of micro-code-based, cross-memory communications and it relies on the same Queued Direct I/O (QDIO) protocol used by the OSA-Express.

With HiperSockets there is no need for external I/O adapters, OSA(-Express) interfaces, or cables for realizing intra-LPAR and inter-LPAR TCP/IP networking within the same mainframe. Given its reliance on QDIO, HiperSockets is also sometimes referred to as 'internal QDIO' (iQDIO), where the 'internal' prefix emphasizes that HiperSockets are possible only inside a zSeries mainframe.

HiperSockets is available on z800, z900, z890, and z990 mainframes. They are supported by the following zSeries

operating systems (at the stated release and greater): z/OS 1.2, z/OS.e 1.3, z/VM 4.2, VSE/ESA 2.7, Linux for zSeries (64-bit mode), and Linux for S/390 (31-bit mode). In the case of z/VM, HiperSockets can also be gainfully used to provide near zero-delay TCP/IP communications between 'guest' OSs running within the same z/VM LPAR. This is particularly germane when you realize that it is possible to run thousands of Linux server images with one z/VM LPAR.

With HyperSockets you can now have memory-speed, TCP/IP networking among all these Linux servers, without any cables – and with performance enhancing 'jumbo' frame sizes in the 64KB range. HiperSockets is thus a highly persuasive argument for mainframe-based Linux server consolidation. When it comes to server consolidation, HiperSockets delivers unprecedented networking performance while at the same time dramatically reducing overall networking complexity. It is indeed a rare win-win technology, with no downside whatsoever, in the context of server farm rationalization.

With z/OS, there is also a HiperSockets Accelerator feature that can be used to optimize and streamline communications between servers using HiperSockets and external networks attached to OSA-Express interfaces. The HiperSockets Accelerator uses a dedicated TCP/IP stack to act as a high-speed traffic concentration point between OSA-Express adapters and internal HiperSockets LANs. At this juncture it should also be noted that HiperSockets can be used, in certain select TCP/IP-specific situations, by Base Sysplex and Parallel Sysplex customers as an alternative to IBM's Cross system Coupling Facility (XCF).

TWO DISTINCT VERSIONS, ALREADY

Once the potential of using HiperSockets to interconnect Linux server images running within one z/VM LPAR is appreciated, it is easy to see why IBM tends to describe HiperSockets as an any-to-any, TCP/IP-based connectivity mechanism between servers occurring within a zSeries mainframe Central Electronic

Complex (CEC) – though substituting OSs in place of servers tends to make this definition even clearer and easier to remember. Another way to think of HiperSockets is as an internal LAN that lets you interconnect any and all TCP/IP stacks implemented within a zSeries mainframe – including multiple stacks that occur within the same LPAR.

When you get to the TCP/IP stack level, variations in HiperSockets implementations, based on the type of zSeries machine involved, come into play. As of April 2004, with the introduction of the new entry-level z890, with the exciting new Application Assist Processors (zAAPs), there are now two distinct classes of HiperSockets. The older generation zSeries machines, ie the z900s and z800s, support up to four HiperSockets LANs per mainframe, while the newer machines, ie the z990s and z890s, support up to 16 HiperSockets LANs.

Per standard mainframe I/O conventions, each HiperSockets LAN is identified by a unique Channel Path Identifier (CHPID) number. HiperSockets-based communications take place in the context of device numbers specified relative to a CHPID.

In the case of z900s and z800s, it is possible, as a rough rule-of-thumb, to support up to 1,024 TCP/IP stacks, per mainframe, across all four HiperSockets LANs. These four HiperSockets LANs can support up to 4,000 IP addresses – which, however, have to include Virtual IP Addresses (VIPA) and dynamic VIPA (DVIPA) addresses assigned to a TCP/IP stack. The z990s and z890s, with their 16 HiperSockets LANs per machine, can support a minimum of 4,096 TCP/IP stacks per machine (and, hopefully, up to 16,000 IP addresses, though IBM does not specify this limit).

IBM'S TCP/IP STACK LIMITS NOT APPLICABLE TO Z/OS

The 1,024 and 4,096 TCP/IP stack limits quoted above from the IBM literature, in reality, are easy-to-remember placeholders, as opposed to being hard-and-fast technical parameters. To fully understand the stack limitations of HiperSockets one

needs to understand how I/O device addresses (per CHPID) are used when it comes to TCP/IP stacks. In the case of the 'older' machines, HiperSockets supports up to 3,072 devices across the four CHPIDs. This number is fixed and cannot be exceeded.

In the case of z/VM and Linux, you need three HiperSockets device addresses per TCP/IP stack connection. The 1,024 stack limit for the older machines is thus derived by dividing the 3,072 device address limit by 3; ie 3,072/3 = 1,024. z/OS uses HiperSockets device addresses in a different way. z/OS, on an LPAR basis, uses one device address per TCP/IP stack within an LPAR – as well as two additional stack control addresses per LPAR, independent of the numbers of stacks. Thus it would be possible to have eight TCP/IP stacks within a single z/OS LPAR and use only 10 HiperSockets device addresses: eight for the eight stacks and two for controlling all eight stacks.

Given this z/OS device address allocation, it would indeed be possible to have more than 1,024 TCP/IP stacks, across four HiperSockets LANs, with z/OS. In addition, with z/OS, it is theoretically possible for some of the HiperSockets device addresses to be shared across different LPARs. The bottom line here is that one should always remember that the 1,024 and 4,096 TCP/IP stack limits quoted for HiperSockets depend on the type of OS being used and that one should consult IBM to get precise numbers, for a specific configuration, if you think you're going to be bumping up against these relatively high limits.

BOTTOM LINE

HiperSockets make the unparalleled 'virtualization' (eg LPARs and Linux server images) capabilities of zSeries machines that much more compelling. HiperSockets, by eliminating the need for conventional communications equipment and communications ports, without question minimizes complexity and cost, while at the same time maximizing reliability and availability. It also offers a level of near zero-latency, high-

speed TCP/IP communication not possible using conventional networking equipment, particularly given that it supports extremely large frame sizes. It is, without doubt, the optimum way to realize zSeries-based server consolidation, especially when trying to integrate large numbers of Linux server images.

Until April 2004, HiperSockets was available only on zSeries machines. The server consolidation potential of HiperSockets was too great, however, to be limited just to mainframes, particularly with IBM now aggressively promoting LPAR and Linux on its pSeries and iSeries machines. Thus, HiperSockets, now referred to as 'Virtual Ethernet', is include as a part of IBM's new 'Virtualization Engine' technology that will, in time, be available on all IBM eServer systems. This, rather than in any way detracting from the significance of HiperSockets *vis-à-vis* mainframes, highlights the value and potential of this innovative TCP/IP methodology when it comes to the future of virtualized computing.

Anura Gurugé
Strategic Consultant (USA)

@FTPGET edit macro

Over the years I have found it convenient to edit text files on the mainframe using ISPF edit. Having more and more distributed platform responsibilities with cross-platform applications, moving source back and forth became a time-consuming activity. So I wrote a pair of edit macros to make my life a little easier. I hope they can simplify your life too.

@FTPGET will present a small pop-up to GET a file from any FTP server and insert it into the current edit session. If the edit session is empty it will insert the new data at the beginning of the edit session. If the edit session already contains data, the

retrieved file will be inserted after all the existing data. The @FTPGET edit macro also supports ‘A’ and ‘B’ line commands to specify whether you want the new data inserted After or Before a specific line.

@FTPGET will capture the FTP output and present it in browse only if the FTP session has a problem. The macro is fully self-contained and needs to be placed in a PDS in the TSO SYSEXEC concatenation. It could also be placed in SYSPROC, but you will have to delete the first line so you have the required REXX comment on the first line.

FTPGET

```
/*********************************************************************/
/*                               REXX                                */
/*********************************************************************/
/* Purpose: Edit Macro to FTP GET a file into an edit session        */
/*-------------------------------------------------------------------*/
/* Syntax:  @FTPGET ftpfile ftpaddr                                  */
/*-------------------------------------------------------------------*/
/* Parms: ftpfile    - Destination directory on target host          */
/*        ftpaddr    - Target host address for receiving the file    */
/*                                                                   */
/* Notes: The file will be appended to the end unless an 'A' or 'B'  */
/*        line command it found.  If 'A' is found the file will be   */
/*        inserted after the line command.  If 'B' is found the      */
/*        file will be inserted before the line command.             */
/*                                                                   */
/*********************************************************************/
/*                      Change Log                                   */
/*                                                                   */
/*********** @REFRESH BEGIN START    2004/03/06 13:16:32 *************/
/* Standard housekeeping activities                                  */
/*********************************************************************/
call time 'r'
parse arg parms
signal on syntax name trap
signal on failure name trap
signal on novalue name trap
probe = 'NONE'
modtrace = 'NO'
modspace = ''
call stdentry 'DIAGMSGS'
module = 'MAINLINE'
push trace() time('L') module 'From:' 0 'Parms:' parms
```

```
if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
call modtrace 'START' Ø
/*********************************************************************/
/* Set local esoteric names                                          */
/*********************************************************************/
@vio   = 'VIO'
@sysda = 'SYSDA'
/*********** @REFRESH END   START    2ØØ4/Ø3/Ø6 13:16:32 *************/
/* Establish Edit Macro environment, Get parms and get dsn and mem   */
/*********************************************************************/
call isrwrap "MACRO (FTPFILE FTPADDR) NOPROCESS"
call isrwrap "(DSN) = DATASET"
call isrwrap "(MEM) = MEMBER"
call isrwrap "(FTPLRECL) = LRECL"
ftplrecl = strip(ftplrecl,'L',Ø)
/*********************************************************************/
/* Allow the A and B line commands to identify After and Before      */
/*********************************************************************/
PRC = isrwrap(2Ø "PROCESS DEST")
select
   when PRC = 8  then dest = '.ZLAST'
   when PRC = 2Ø then dest = '.ZFIRST'
   otherwise dest = '.ZDEST'
end
/*********************************************************************/
/* Put the FTPFILE and FTPADDR in the profile pool if provided       */
/*********************************************************************/
if ftpfile <> '' then call ispwrap "VPUT (FTPFILE) PROFILE"
if ftpaddr <> '' then call ispwrap "VPUT (FTPADDR) PROFILE"
/*********************************************************************/
/* Check to see whether the profile variables exist                  */
/*********************************************************************/
call ispwrap 8 "VGET (FTPFILE) PROFILE"
call ispwrap 8 "VGET (FTPADDR) PROFILE"
/*********************************************************************/
/* Build a temporary panel                                           */
/*********************************************************************/
ftp.1  = ")ATTR                                                 "
ftp.2  = " @ TYPE(DT)                                           "
ftp.3  = " # TYPE(INPUT) CAPS(OFF) COLOR(GREEN)                 "
ftp.4  = " $ TYPE(INPUT) CAPS(OFF) INTENS(NON)                  "
ftp.5  = ")BODY EXPAND(//) WINDOW(6Ø,7)                         "
ftp.6  = "+FTP Site can be full IP address or DNS Name          "
ftp.7  = "@Enter FTP Site ID  : #Z                              "
ftp.8  = "@Enter FTP UserID   : #Z        +                     "
ftp.9  = "@Enter FTP Password : $Z       +                      "
ftp.1Ø = "@Enter FTP GET File : #Z                              "
ftp.11 = "@LRECL for upload   : _Z   +(defaults to Edit DSN)"
ftp.12 = "@WRAPtext           : _Z   +(%YES+or%NO+)             "
ftp.13 = ")INIT                                                 "
```

```
ftp.14 = " .ZVARS = '(FTPADDR FTPUSER FTPPASS FTPFILE +     "
ftp.15 = "            FTPLRECL FTPWRAP)'                    "
ftp.16 = " IF (&FTPUSER = &Z)                               "
ftp.17 = "     &FTPUSER = 'ANONYMOUS'                       "
ftp.18 = " IF (&FTPPASS = &Z)                               "
ftp.19 = "     &FTPPASS = &ZUSER                            "
ftp.2Ø = " IF (&FTPFILE = &Z)                               "
ftp.21 = "     &FTPFILE = '/'                               "
ftp.22 = " IF (&FTPLRECL = &Z)                              "
ftp.23 = "     &FTPLRECL = '8Ø'                             "
ftp.24 = " IF (&FTPWRAP = &Z)                               "
ftp.25 = "     &FTPWRAP = 'NO'                              "
ftp.26 = ")PROC                                             "
ftp.27 = " VER (&FTPADDR,NB)                                "
ftp.28 = " VER (&FTPUSER,NB)                                "
ftp.29 = " VER (&FTPPASS,NB)                                "
ftp.3Ø = " VER (&FTPFILE,NB)                                "
ftp.31 = " VER (&FTPLRECL,NB,NUM)                           "
ftp.32 = " VER (&FTPWRAP,NB,LIST,YES,NO)                    "
ftp.33 = ")END                                              "
/*********************************************************************/
/* Display the Dynamic Panel                                         */
/*********************************************************************/
call popdyn 'FTP' 4 'Enter' execname 'Parameters'
/*********************************************************************/
/* VPUT the FTPFILE and FTPADDR                                      */
/*********************************************************************/
call ispwrap "VPUT (FTPFILE) PROFILE"
call ispwrap "VPUT (FTPADDR) PROFILE"
/*********************************************************************/
/* Allocate a temporary dataset to hold the copy                     */
/*********************************************************************/
ddname = '@FTP'random()
dsname = userid()'.'execname'.'ddname'.FILE'
/*********************************************************************/
/* Stage the FTP commands in stem variables                          */
/*********************************************************************/
input.1 =  ftpuser ftppass
input.2 = "LOCSITE LRECL="ftplrecl "BLKSIZE=Ø"
if ftpwrap = 'YES' then
    input.3 = "LOCSITE WRAP"
else
    input.3 = "LOCSITE NOWRAP"
/* Determine whether this is a Unix file or an MVS DSN             */
if substr(ftpfile,1,1) = '/' then
    do
     input.4 = "get" ftpfile "'"dsname"'"
     input.5 = "quit"
    end
else
```

```
/* If it is a DSN, determine whether there is a member or not       */
    do
     if pos('(',ftpfile) <> Ø then
        do
         parse var ftpfile fromdsn '(' frommem ')'
         input.4 = "cd '"fromdsn"'"
         input.5 = "get" frommem "'"dsname"'"
         input.6 = "quit"
        end
     else
        do
         input.4 = "get '"ftpfile"' '"dsname"'"
         input.5 = "quit"
        end
    end
/*********************************************************************/
/* Lock the screen                                                   */
/*********************************************************************/
call lock input.4
/*********************************************************************/
/* Copy the lines to the temporary INPUT dataset                     */
/*********************************************************************/
call viodd 'INPUT'
/*********************************************************************/
/* Allocate a temporary dataset to hold FTP OUTPUT                   */
/*********************************************************************/
call tsotrap "ALLOC F(OUTPUT) NEW REUSE UNIT(SYSDA) SPACE(1 1) CYL"
/*********************************************************************/
/* FTP the File                                                      */
/*********************************************************************/
address TSO "FTP" ftpaddr "(EXIT"
FTPRC = RC
call unlock
/* If there is an FTP error, browse the OUTPUT dataset               */
if FTPRC <> Ø then
    do
     call msg 'FTP error, RC='FTPRC 'Review the FTP output'
     call brwsdd 'OUTPUT'
     call tsotrap "FREE F(INPUT)"
     call rcexit FTPRC 'FTP from' ftpaddr 'failed'
    end
else
    do
     zedlmsg = ftpfile 'successfully loaded from' ftpaddr
     address ISPEXEC "SETMSG MSG(ISRZØØØ) COND"
    end
/* Move the FTP'd dataset into the current edit session              */
call isrwrap "MOVE '"dsname"' AFTER" dest
call isrwrap "UP MAX"
/* Clean up temporary FTP INPUT and OUTPUT                           */
```

```
call tsotrap "FREE F(INPUT)"
call tsotrap "FREE F(OUTPUT)"
/* Shutdown                                                          */
shutdown: nop
/* Shutdown/clean-up processing                                      */
          EXITRC = 1
/* Shutdown message and terminate                                    */
          call stdexit time('e')
/* 28 Internal Subroutines provided in @FTPGET                       */
/* Last Subroutine REFRESH was 16 Apr 2004 17:39:13                  */
/* RCEXIT   - Exit on non-zero return codes                          */
/* TRAP     - Issue a common trap error message using rcexit         */
/* ERRMSG   - Build common error message with failing line number    */
/* STDENTRY - Standard Entry logic                                   */
/* STDEXIT  - Standard Exit logic                                    */
/* MSG      - Determine whether to SAY or ISPEXEC SETMSG the message */
/* DDCHECK  - Determine whether a required DD is allocated           */
/* QDSN     - Make sure there are only one set of quotes             */
/* TEMPMEM  - EXECIO a stem into a TEMP PDS                          */
/* BRWSDD   - Invoke ISPF Browse on any DD                           */
/* ISPWRAP  - Wrapper for ISPF commands                              */
/* ISRWRAP  - Wrapper for ISPF Edit commands                         */
/* TSOTRAP  - Capture the output from a TSO command in a stem        */
/* WAIT     - Wait for a specified number of seconds                 */
/* SETBORD  - Set the ISPF Pop-up active frame border color          */
/* LOCK     - Put up a pop-up under foreground ISPF during long waits*/
/* UNLOCK   - Unlock from a pop-up under foreground ISPF             */
/* PANDSN   - Create a unique PDS(MEM) name for a dynamic panel      */
/* POPDYN   - Addpop a Dynamic Panel                                 */
/* SAYDD    - Print messages to the requested DD                     */
/* JOBINFO  - Get job related data from control blocks               */
/* PTR      - Pointer to a storage location                          */
/* STG      - Return the data from a storage location                */
/* VIODD    - EXECIO a stem into a sequential dataset                */
/* MODTRACE - Module Trace                                           */
/* RCEXIT   - Exit on non-zero return codes                          */
/* EXITRC   - Return code to exit with (if non zero)                 */
/* ZEDLMSG  - Message text for it with for non zero EXITRC's         */
rcexit: parse arg EXITRC zedlmsg
         if EXITRC <> 0 then
            do
             trace 'o'
/* If execution environment is ISPF then VPUT ZISPFRC                */
             if execenv = 'TSO' | execenv = 'ISPF' then
                do
                 if ispfenv = 'YES' then
                    do
                     zispfrc = EXITRC
/* Does not call ISPWRAP to avoid obscuring error message modules    */
                     address ISPEXEC "VPUT (ZISPFRC)"
```

```
                end
             end
/* If a message is provided, wrap it in date, time, and EXITRC       */
          if zedlmsg <> '' then
             do
              zedlmsg = time('L') execname zedlmsg 'RC='EXITRC
              call msg zedlmsg
             end
/* Write the contents of the Parentage Stack                         */
           stacktitle = 'Parentage Stack Trace ('queued()' entries):'
/* Write to MSGDD if background and MSGDD exists                     */
          if tsoenv = 'BACK' then
             do
              if subword(zedlmsg,9,1) = msgdd then
                 do
                  say zedlmsg
                  signal shutdown
                 end
               else
                 do
                  call saydd msgdd 1 zedlmsg
                  call saydd msgdd 1 stacktitle
                 end
             end
          else
/* Write to the ISPF Log if foreground                               */
             do
              zerrlm = zedlmsg
              address ISPEXEC "LOG MSG(ISRZØØ3)"
              zerrlm = center(' 'stacktitle' ',78,'-')
              address ISPEXEC "LOG MSG(ISRZØØ3)"
             end
/* Unload the Parentage Stack                                        */
          do queued()
             pull stackinfo
             if tsoenv = 'BACK' then
                do
                 call saydd msgdd Ø stackinfo
                end
             else
                do
                 zerrlm = stackinfo
                 address ISPEXEC "LOG MSG(ISRZØØ3)"
                end
          end
/* Put a terminator in the ISPF Log for the Parentage Stack          */
          if tsoenv = 'FORE' then
             do
              zerrlm = center(' 'stacktitle' ',78,'-')
              address ISPEXEC "LOG MSG(ISRZØØ3)"
```

```
               end
/* Signal SHUTDOWN.  SHUTDOWN label MUST exist in the program        */
             signal shutdown
            end
         else
            return
/* TRAP     - Issue a common trap error message using rcexit         */
/* PARM     - N/A                                                    */
trap: traptype = condition('C')
       if traptype = 'SYNTAX' then
          msg = errortext(RC)
       else
          msg = condition('D')
       trapline = strip(sourceline(sigl))
       msg = traptype 'TRAP:' msg', Line:' sigl '"'trapline'"'
       call rcexit 666 msg
/* ERRMSG   - Build common error message with failing line number    */
/* ERRLINE  - The failing line number passed by caller from SIGL     */
/* TEXT     - Error message text passed by caller                    */
errmsg: nop
         parse arg errline text
         return 'Error on statement' errline',' text
/* STDENTRY - Standard Entry logic                                   */
/* MSGDD    - Optional MSGDD used only in background                 */
stdentry: module = 'STDENTRY'
           if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
           parse arg sparms
           push trace() time('L') module 'From:' sigl 'Parms:' sparms
           arg msgdd
           parse upper source . . execname . execdsn . . execenv .
/* Start-up values                                                   */
           EXITRC = Ø
           MAXRC = Ø
           ispfenv = 'NO'
           popup = 'NO'
           lockpop = 'NO'
           headoff = 'NO'
           hcreator = 'NO'
           keepstack = 'NO'
           lpar = mvsvar('SYSNAME')
           zedlmsg = 'Default shutdown message'
/* Determine environment                                             */
           if substr(execenv,1,3) <> 'TSO' & execenv <> 'ISPF' then
              tsoenv = 'NONE'
           else
              do
               tsoenv = sysvar('SYSENV')
               signal off failure
              "ISPQRY"
               ISPRC = RC
```

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

```
/* If there is a STEPLIB, write the STEPLIB DSN MSGDD                */
                    if listdsi('STEPLIB' 'FILE') = Ø then
                       do
                        steplibs = dddsns('STEPLIB')
                        call saydd msgdd Ø 'STEPLIB executables loaded',
                            'from' word(dddsns,1)
                        if dddsns('STEPLIB') > 1 then
                           do
                            do stl=2 to steplibs
                               call saydd msgdd Ø copies(' ',31),
                                    word(dddsns,stl)
                            end
                           end
                       end
                 end
              end
/* If foreground, save ZFKA and turn off the FKA display             */
           else
              do
               fkaset = 'OFF'
               call ispwrap "VGET (ZFKA) PROFILE"
               if zfka <> 'OFF' & tsoenv = 'FORE' then
                  do
                   fkaset = zfka
                   fkacmd = 'FKA OFF'
                   call ispwrap "CONTROL DISPLAY SAVE"
                  call ispwrap "DISPLAY PANEL(ISPBLANK) COMMAND(FKACMD)"
                   call ispwrap "CONTROL DISPLAY RESTORE"
                  end
              end
            pull tracelvl . module . sigl . sparms
            call modtrace 'STOP' sigl
            interpret 'trace' tracelvl
            return
/* STDEXIT  - Standard Exit logic                                    */
/* ENDTIME  - Elapsed time                                           */
/* Note: Caller must set KEEPSTACK if the stack is valid             */
stdexit: module = 'STDEXIT'
           if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
           parse arg sparms
           push trace() time('L') module 'From:' sigl 'Parms:' sparms
           call modtrace 'START' sigl
           arg endtime
           endmsg = execname 'ended' date() time() format(endtime,,1)
/* if MAXRC is greater then EXITRC then set EXITRC to MAXRC          */
           if MAXRC > EXITRC then EXITRC = MAXRC
           endmsg = endmsg 'on' lpar 'RC='EXITRC
           if tsoenv = 'BACK' & sysvar('SYSNEST') = 'NO' &,
              headoff = 'NO' then
              do
```

```
              say
              say jobname center(' 'endmsg' ',61,'-') jobtype jobnum
/* Make sure this isn't a MSGDD missing error then log to MSGDD      */
              if msgdd <> '' & subword(zedlmsg,9,1) <> msgdd then
                 do
                  call saydd msgdd 1 execname 'ran in' endtime 'seconds'
                  call saydd msgdd Ø endmsg
                 end
              end
/* If foreground, reset the FKA if necessary                         */
           else
              do
               if fkaset <> 'OFF' then
                  do
                   fkafix = 'FKA'
                   call ispwrap "CONTROL DISPLAY SAVE"
                   call ispwrap "DISPLAY PANEL(ISPBLANK) COMMAND(FKAFIX)"
                   if fkaset = 'SHORT' then
                      call ispwrap "DISPLAY PANEL(ISPBLANK)",
                                   "COMMAND(FKAFIX)"
                   call ispwrap "CONTROL DISPLAY RESTORE"
                  end
              end
/* Clean up the temporary HEADING table                              */
           if ispfenv = 'YES' & hcreator = 'YES' then
              call ispwrap "TBEND" htable
/* Remove STDEXIT and MAINLINE Parentage Stack entries, if there     */
           call modtrace 'STOP' sigl
           if queued() > Ø then pull . . module . sigl . sparms
           if queued() > Ø then pull . . module . sigl . sparms
           if tsoenv = 'FORE' & queued() > Ø & keepstack = 'NO' then
              pull . . module . sigl . sparms
/* if the Parentage Stack is not empty, display its contents         */
           if queued() > Ø & keepstack = 'NO' then
              do
               say queued() 'Leftover Parentage Stack Entries:'
               say
               do queued()
                  pull stackundo
                  say stackundo
               end
               EXITRC = 1
              end
/* Exit                                                              */
           exit(EXITRC)
/* MSG      - Determine whether to SAY or ISPEXEC SETMSG the message */
/* ZEDLMSG  - The long message variable                              */
msg: module = 'MSG'
     parse arg zedlmsg
     if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
```

```
      parse arg sparms
      push trace() time('L') module 'From:' sigl 'Parms:' sparms
      call modtrace 'START' sigl
/* If this is background or OMVS use SAY                              */
      if tsoenv = 'BACK' | execenv = 'OMVS' then
         say zedlmsg
      else
/* If this is foreground and ISPF is available, use SETMSG            */
         do
          if ispfenv = 'YES' then
/* Does not call ISPWRAP to avoid obscuring error message modules     */
              address ISPEXEC "SETMSG MSG(ISRZØØØ)"
          else
             say zedlmsg
         end
      pull tracelvl . module . sigl . sparms
      call modtrace 'STOP' sigl
      interpret 'trace' tracelvl
      return
/* DDCHECK  - Determine whether a required DD is allocated            */
/* DD       - DDNAME to confirm                                       */
ddcheck: module = 'DDCHECK'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg dd
          dderrmsg = 'OK'
          LRC = listdsi(dd "FILE")
/* Allow sysreason=3 to verify SYSOUT DD statements                   */
          if LRC <> Ø & strip(sysreason,'L',Ø) <> 3 then
             do
              dderrmsg = errmsg(sigl 'Required DD' dd 'is missing')
              call rcexit LRC dderrmsg sysmsglvl2
             end
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return
/* QDSN     - Make sure there is only one set of quotes               */
/* QDSN     - The DSN                                                 */
qdsn: module = 'QDSN'
       if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
       parse arg sparms
       push trace() time('L') module 'From:' sigl 'Parms:' sparms
       call modtrace 'START' sigl
       parse arg qdsn
       qdsn = "'"strip(qdsn,"B","'")"'"
       pull tracelvl . module . sigl . sparms
       call modtrace 'STOP' sigl
```

```
       interpret 'trace' tracelvl
       return qdsn
/* TEMPMEM  - EXECIO a stem into a TEMP PDS                          */
/* TEMPMEM  - The member to create                                   */
tempmem: module = 'TEMPMEM'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg tempmem
/* Create a unique DD name                                           */
          if length(tempmem) < 7 then
             tempdd = '$'tempmem'$'
          else
             tempdd = '$'substr(tempmem,2,6)'$'
/* If TEMPDD exists, then FREE it                                    */
          if listdsi(tempdd 'FILE') = Ø then "FREE F("tempdd")"
/* Generate the TEMPDSN and carve out the DSN                        */
          tempdsn = pandsn(tempmem)
          parse var tempdsn temppds '(' .
/* ALLOCATE the TEMP DD and member                                   */
          call tsotrap "ALLOC F("tempdd") DA("qdsn(tempdsn)") NEW",
                       "LRECL(8Ø) BLKS(Ø) DIR(1) SPACE(1) CATALOG",
                       "UNIT("@sysda") RECFM(F B)"
/* Write the STEM to the TEMP DD                                     */
          call tsotrap 'EXECIO * DISKW' tempdd '(STEM' tempmem'. FINIS'
/* DROP the stem variable                                            */
          interpret 'drop' tempmem'.'
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return tempdd
/* BRWSDD   - Invoke ISPF Browse on any DD                           */
/* BRWSDD   - Any DD to browse                                       */
brwsdd: module = 'BRWSDD'
         if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
         arg brwsdd
         if brwsdd = '' then call rcexit 9Ø 'Browse DD missing'
         call ispwrap "LMINIT DATAID(DATAID) DDNAME("brwsdd")"
/* Browse the VIO dataset                                            */
         call ispwrap "BROWSE DATAID("dataid")"
/* FREE and DELETE the VIO dataset                                   */
         call ispwrap "LMFREE DATAID("dataid")"
         call tsotrap "FREE F("brwsdd")"
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
```

```
         return
/* ISPWRAP  - Wrapper for ISPF commands                              */
/* VALIDRC  - Optional valid RC from the ISPF command, defaults to Ø */
/* ISPPARM  - Valid ISPF command                                     */
ispwrap: module = 'ISPWRAP'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          parse arg ispparm
          zerrlm = 'NO ZERRLM'
/* If the optional valid_rc parm is present use it, if not assume Ø  */
          parse var ispparm valid_rc isp_cmd
          if datatype(valid_rc,'W') = Ø then
             do
              valid_rc = Ø
              isp_cmd = ispparm
             end
          address ISPEXEC isp_cmd
          IRC = RC
/* If RC = Ø then return                                             */
          if IRC <= valid_rc then
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return IRC
             end
          else
             do
              perrmsg = errmsg(sigl 'ISPF Command:')
              call rcexit IRC perrmsg isp_cmd strip(zerrlm)
             end
/* ISRWRAP  - Wrapper for ISPF Edit commands                         */
/* VALIDRC  - Optional valid RC from the ISPF command, defaults to Ø */
/* ISRPARM  - Valid ISPF Edit command                                */
isrwrap: module = 'ISRWRAP'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          parse arg isrparm
/* If the optional valid_rc parm is present use it, if not assume Ø  */
          parse var isrparm valid_rc isr_cmd
          if datatype(valid_rc,'W') = Ø then
             do
              valid_rc = Ø
              isr_cmd = isrparm
             end
          parse var isr_cmd isr_verb .
```

```
          address ISREDIT isr_cmd
          ERC = RC
/* If RC = Ø then return                                              */
          if ERC <= valid_rc then
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return ERC
             end
          else
             do
              if isr_verb = 'MACRO' & ERC = 2Ø then
                 do
                  call rcexit ERC 'is an Edit Macro and not valid to',
                              'run outside of ISPF Edit'
                 end
              else
                 do
                  rerrmsg = errmsg(sigl 'ISPF Edit Command:')
                  call rcexit ERC rerrmsg isr_cmd
                 end
             end
/* TSOTRAP  - Capture the output from a TSO command in a stem        */
/* VALIDRC  - Optional valid RC, defaults to zero                    */
/* TSOPARM  - Valid TSO command                                      */
tsotrap: module = 'TSOTRAP'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          parse arg tsoparm
/* If the optional valid_rc parm is present use it, if not assume Ø  */
          parse var tsoparm valid_rc tso_cmd
          if datatype(valid_rc,'W') = Ø then
             do
              valid_rc = Ø
              tso_cmd = tsoparm
             end
          call outtrap 'tsoout.'
          tsoline = sigl
          address TSO tso_cmd
          CRC = RC
          call outtrap 'off'
/* If RC = Ø then return                                              */
          if CRC <= valid_rc then
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
```

```
             return CRC
            end
          else
            do
             trapmsg = center(' TSO Command Error Trap ',78,'-')
             terrmsg = errmsg(sigl 'TSO Command:')
/* If RC <> Ø then format output depending on environment            */
             if tsoenv = 'BACK' | execenv = 'OMVS' then
               do
                say trapmsg
                do c=1 to tsoout.Ø
                   say tsoout.c
                end
                say trapmsg
                call rcexit CRC terrmsg tso_cmd
               end
             else
/* If this is foreground and ISPF is available, use the ISPF LOG    */
               do
                if ispfenv = 'YES' then
                  do
                   zedlmsg = trapmsg
/* Does not call ISPWRAP to avoid obscuring error message modules   */
                    address ISPEXEC "LOG MSG(ISRZØØØ)"
                    do c=1 to tsoout.Ø
                       zedlmsg = tsoout.c
                       address ISPEXEC "LOG MSG(ISRZØØØ)"
                    end
                    zedlmsg = trapmsg
                    address ISPEXEC "LOG MSG(ISRZØØØ)"
                    call rcexit CRC terrmsg tso_cmd,
                        ' see the ISPF Log (Option 7.5) for details'
                  end
                else
                  do
                   say trapmsg
                   do c=1 to tsoout.Ø
                      say tsoout.c
                   end
                   say trapmsg
                   call rcexit CRC terrmsg tso_cmd
                  end
               end
            end
/* WAIT     - Wait for a specified number of seconds                */
/* SECONDS  - Number of seconds to wait                             */
/* WMODE    - Use any value to stop printing batch wait messages    */
wait: module = 'WAIT'
       if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
       parse arg sparms
```

```
       push trace() time('L') module 'From:' sigl 'Parms:' sparms
       call modtrace 'START' sigl
       arg seconds wmode
       if datatype(seconds,'W') = Ø then seconds = 1Ø
       RC = syscalls('ON')
/* If foreground ISPF lock the screen                                  */
       if tsoenv = 'FORE' & ispfenv = 'YES' then
          call lock seconds 'second wait was requested'
/* If background, report the wait time                                 */
       if tsoenv = 'BACK' & wmode = '' then
          call saydd msgdd Ø seconds 'second wait was requested'
/* Call USS SLEEP                                                      */
       address SYSCALL "SLEEP" seconds
/* If foreground ISPF lock the screen                                  */
       if tsoenv = 'FORE' & ispfenv = 'YES' then
          call unlock
       RC = syscalls('OFF')
       pull tracelvl . module . sigl . sparms
       call modtrace 'STOP' sigl
       interpret 'trace' tracelvl
       return
/* SETBORD  - Set the ISPF Popup active frame border color            */
/* COLOR    - Colour for the Active Frame Border                      */
setbord: module = 'SETBORD'
         if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
         arg color
/* Parse and validate colour                                           */
         if color = '' then color = 'YELLOW'
/* Build a temporary panel                                             */
         ispopt11.1=")BODY                        "
         ispopt11.2="%Command ===>_ZCMD        + "
         ispopt11.3=")INIT                        "
         ispopt11.4="&ZCMD = ' '                  "
         ispopt11.5="VGET (COLOR) SHARED          "
         ispopt11.6="&ZCOLOR = &COLOR             "
         ispopt11.7=".RESP = END                  "
         ispopt11.8=")END                         "
/* Allocate and load the Dynamic Panel                                 */
         setdd = tempmem('ISPOPT11')
/* LIBDEF the DSN, VPUT @TFCOLOR and run CUAATTR                       */
         call ispwrap "LIBDEF ISPPLIB LIBRARY ID("setdd") STACK"
         call ispwrap "VPUT (COLOR) SHARED"
         call ispwrap "SELECT PGM(ISPOPT) PARM(ISPOPT11)"
         call ispwrap "LIBDEF ISPPLIB"
         call tsotrap "FREE F("setdd") DELETE"
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
```

```
          interpret 'trace' tracelvl
          return
/* LOCK     - Put up a popup under foreground ISPF during long waits */
/* LOCKMSG  - Message for the pop-up screen                           */
lock: module = 'LOCK'
       if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
       parse arg sparms
       push trace() time('L') module 'From:' sigl 'Parms:' sparms
       call modtrace 'START' sigl
       parse arg lockmsg
       if lockmsg = '' then lockmsg = 'Please be patient'
       if tsoenv = 'FORE' then
          do
/* Use the length of the lockmsg to determine the pop-up size        */
           if length(lockmsg) < 76 then
              locklen = length(lockmsg) + 2
           else
              locklen = 77
           if locklen <= 1Ø then locklen = 1Ø
/* Build a temporary panel                                           */
           lock.1 = ")BODY EXPAND(//) WINDOW("locklen",1)"
           lock.2 = "%&LOCKMSG                    "
           lock.3 = ")END                         "
/* Lock the screen and put up a pop-up                               */
           call ispwrap "CONTROL DISPLAY LOCK"
           call popdyn 'LOCK' 8 execname 'Please be patient'
           lockpop = 'YES'
          end
       pull tracelvl . module . sigl . sparms
       call modtrace 'STOP' sigl
       interpret 'trace' tracelvl
       return
/* UNLOCK   - Unlock from a pop-up under foreground ISPF             */
/* PARM     - N/A                                                    */
unlock: module = 'UNLOCK'
         if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
         if tsoenv = 'FORE' then
             do
              if lockpop = 'YES' then
                 do
                  call ispwrap "REMPOP"
                  lockpop = 'NO'
                 end
              if popup = 'YES' then
                 do
                  call setbord 'BLUE'
                  call ispwrap "REMPOP"
```

```
                popup = 'NO'
               end
            end
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return
/* PANDSN   - Create a unique PDS(MEM) name for a dynamic panel      */
/* PANEL    - Dynamic panel name                                     */
pandsn: module = 'PANDSN'
         if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
         arg panel
         unique = right(time('s'),7,0)||'('||panel||')'
         pandsn = userid()||'.'||execname||'.'||panel||'.T'||unique
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return pandsn
/* POPDYN   - Addpop a Dynamic Panel                                 */
/* DYN      - Dynamic panel name                                     */
/* DYNROW   - Default row for ADDPOP, defaults to 1                  */
/* DYNMSG   - ADDPOP Window title                                    */
popdyn: module = 'POPDYN'
         if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
         parse arg dyn dynrow dynmsg
/* Set the default ADDPOP row location                               */
         if dynrow = '' then dynrow = 1
/* Set the default ADDPOP window title to the current exec name      */
         if dynmsg = '' then dynmsg = execname
/* Check if the RETURN option is specified in the DYNMSG             */
         dynreturn = 'NO'
         if word(dynmsg,1) = 'RETURN' then
            parse var dynmsg dynreturn dynmsg
/* Allocate and load the Dynamic Panel                               */
         dyndd = tempmem(dyn)
/* LIBDEF the POPDYN panel                                           */
         call ispwrap "LIBDEF ISPPLIB LIBRARY ID("dyndd") STACK"
/* Change the Active Frame Colour                                    */
         call setbord 'YELLOW'
/* set the POPUP variable if this is not a LOCK request             */
         if dyn = 'LOCK' & popup = 'NO' then
            popup = 'NO'
         else
            popup = 'YES'
```

```
/* Put up the pop-up                                                  */
         zwinttl = dynmsg
         call ispwrap "ADDPOP ROW("dynrow")"
         DRC = ispwrap(8 "DISPLAY PANEL("dyn")")
         call ispwrap "LIBDEF ISPPLIB"
         call tsotrap "FREE F("dyndd") DELETE"
/* Change the Active Frame Color                                      */
         call setbord 'BLUE'
/* Determine how to return                                            */
         if dynreturn = 'NO' then
            call rcexit DRC 'terminated by user'
         if dynreturn = 'RETURN' & DRC = 8 then
            do
             call ispwrap "REMPOP"
             popup = 'NO'
            end
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return DRC
/* SAYDD    - Print messages to the requested DD                      */
/* MSGDD    - DDNAME to write messages to                             */
/* MSGLINES - number of blank lines to put before and after           */
/* MESSAGE  - Text to write to the MSGDD                              */
saydd: module = 'SAYDD'
        if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
        parse arg sparms
        push trace() time('L') module 'From:' sigl 'Parms:' sparms
        call modtrace 'START' sigl
        parse arg msgdd msglines message
        if words(msgdd msglines message) < 3 then
           call rcexit 33 'Missing MSGDD or MSGLINES'
        if datatype(msglines) <> 'NUM' then
           call rcexit 34 'MSGLINES must be numeric'
/* If this is not background then bypass                              */
        if tsoenv <> 'BACK' then
           do
            pull tracelvl . module . sigl . sparms
            call modtrace 'STOP' sigl
            interpret 'trace' tracelvl
            return
           end
/* Confirm the MSGDD exists                                           */
        call ddcheck msgdd
/* If a number is provided, add that number of blank lines before     */
/* the message                                                        */
        msgb = 1
        if msglines > Ø then
           do msgb=1 to msglines
              msgline.msgb = ' '
```

```
           end
/* If the linesize is too long break it into multiple lines and      */
/* create continuation records                                       */
        msgm = msgb
        if length(message) > 6Ø & substr(message,1,2) <> '@@' then
           do
            messst = lastpos(' ',message,6Ø)
            messseg = substr(message,1,messst)
            msgline.msgm = date() time() strip(messseg)
            message = strip(delstr(message,1,messst))
            do while length(message) > Ø
               msgm = msgm + 1
               if length(message) > 55 then
                  messst = lastpos(' ',message,55)
               if messst > Ø then
                  messseg = substr(message,1,messst)
               else
                  messseg = substr(message,1,length(message))
               msgline.msgm = date() time() 'CONT:' strip(messseg)
               message = strip(delstr(message,1,length(messseg)))
            end
           end
        else
/* Build print lines. Default strips and prefixes date and timestamp */
/* @BLANK - Blank line, no date and timestamp                        */
/* @      - No stripping, retains leading blanks                     */
/* @@     - No stripping, No date and timestamp                      */
           do
            select
               when message = '@BLANK@' then msgline.msgm = ' '
               when word(message,1) = '@' then
                    do
                     message = substr(message,2,length(message)-1)
                     msgline.msgm = date() time() message
                    end
               when substr(message,1,2) = '@@' then
                    do
                     message = substr(message,3,length(message)-2)
                     msgline.msgm = message
                    end
               otherwise msgline.msgm = date() time() strip(message)
            end
           end
/* If a number is provided, add that number of blank lines after     */
/* the message                                                       */
        if msglines > Ø then
           do msgt=1 to msglines
              msge = msgt + msgm
              msgline.msge = ' '
           end
```

```
/* Write the contents of the MSGLINE stem to the MSGDD               */
        call tsotrap "EXECIO * DISKW" msgdd "(STEM MSGLINE. FINIS"
        drop msgline. msgb msgt msge
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return
/* JOBINFO  - Get job related data from control blocks               */
/* ITEM     - Optional item number desired, default is all           */
jobinfo: module = 'JOBINFO'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg item
/* Chase control blocks                                              */
          tcb      = ptr(54Ø)
          ascb     = ptr(548)
          tiot     = ptr(tcb+12)
          jscb     = ptr(tcb+18Ø)
          ssib     = ptr(jscb+316)
          asid     = c2d(stg(ascb+36,2))
          jobtype  = stg(ssib+12,3)
          jobnum   = strip(stg(ssib+15,5),'L',Ø)
          stepname = stg(tiot+8,8)
          procstep = stg(tiot+16,8)
          program  = stg(jscb+36Ø,8)
          jobdata  = jobtype jobnum stepname procstep program asid
/* Return job data                                                   */
          if item <> '' & (datatype(item,'W') = 1) then
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return word(jobdata,item)
             end
          else
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return jobdata
             end
/* PTR      - Pointer to a storage location                          */
/* ARG(1)   - Storage Address                                        */
ptr: return c2d(storage(d2x(arg(1)),4))
/* STG      - Return the data from a storage location                */
/* ARG(1)   - Location                                               */
/* ARG(2)   - Length                                                 */
stg: return storage(d2x(arg(1)),arg(2))
```

```
/* VIODD    - EXECIO a stem into a sequential dataset                */
/* VIODD    - The member to create                                   */
/* VIOLRECL - The LRECL for the VIODD (defautls to 8Ø)               */
viodd: module = 'VIODD'
        if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
        parse arg sparms
        push trace() time('L') module 'From:' sigl 'Parms:' sparms
        call modtrace 'START' sigl
        arg viodd violrecl viorecfm
        if viodd = '' then call rcexit 88 'VIODD missing'
        if violrecl = '' then violrecl = 8Ø
        if viorecfm = '' then viorecfm = 'F B'
/* If DD exists, FREE it                                            */
        if listdsi(viodd 'FILE') = Ø then
           call tsotrap "FREE F("viodd")"
/* ALLOCATE a VIO DSN                                               */
        call tsotrap "ALLOC F("viodd") UNIT("@vio") SPACE(1 5)",
                     "LRECL("violrecl") BLKSIZE(Ø) REUSE",
                     "RECFM("viorecfm") CYLINDERS"
/* Write the stem variables into the VIO DSN                        */
        call tsotrap "EXECIO * DISKW" viodd "(STEM" viodd". FINIS"
/* DROP the stem variable                                           */
        interpret 'drop' viodd'.'
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return
/* MODTRACE - Module Trace                                          */
/* TRACETYP - Type of trace entry                                   */
/* SIGLINE  - The line number called from                           */
modtrace: if modtrace = 'NO' then return
           arg tracetyp sigline
           tracetyp = left(tracetyp,5)
           sigline = left(sigline,5)
/* Adjust MODSPACE for START                                        */
           if tracetyp = 'START' then
              modspace = substr(modspace,1,length(modspace)+1)
/* Set the trace entry                                              */
           traceline = modspace time('L') tracetyp module sigline sparms
/* Adjust MODSPACE for STOP                                         */
           if tracetyp = 'STOP' then
              modspace = substr(modspace,1,length(modspace)-1)
/* Determine where to write the traceline                           */
           if ispfenv = 'YES' & tsoenv = 'FORE' then
/* Write to the ISPF Log, do not use ISPWRAP here                   */
             do
              zedlmsg = traceline
              address ISPEXEC "LOG MSG(ISRZØØØ)"
             end
           else
```

```
              say traceline
/* SAY to SYSTSPRT                                              */
           return
```

Robert Zenuk
Systems Programmer (USA)

Principles of mainframe TCP/IP management

The wholesale transition of mainframe networking from SNA to IP, which began in earnest around five years ago, is now nearly complete. This iconoclastic conversion, despite the initial misgivings of the SNA community, has been a resounding success. Mainframe mission-critical applications, even those that are SNA-based, still maintain their reputation for up-time and responsiveness, though the SNA-related interactions are now realized across a TCP/IP fabric through a combination of Enterprise Extender (EE), Data Link Switching (DLSw), tn3270(E), and Web-to-host.

Despite IP's triumph, mainframe professionals are, however, still entitled to feel nostalgic, and even aggrieved, when confronted with one crucial aspect of mainframe networking. And that, ironically, has to do with network management! Mainframe-centric IP management and monitoring, at present, can be challenging and confounding – particularly when compared with the consistency and uniformity that characterized NetView-based SNA management in the early 1990s.

SNMP IS NOT ALWAYS THE ANSWER

There are multiple reasons why IP-based mainframe monitoring and management is inherently complex and often confounding. For a start, in contrast to the overall homogeneity of SNA (whether one was dealing with LU2 or LU 6.2), IP networking, even at the mainframe level, will invariably consist of a cocktail

of disparate protocols ranging from FTP to telnet, with routing-related protocols such as OSPF thrown into the middle. Furthermore, SNA, with all its SSCP-related control sessions, was highly controlling and insisted on having its tentacles reach out to every PU. Not so with IP. IP networking eschews control sessions and, compared with SNA, is ultra 'laid-back'. It is possible and common to have connectionless interactions, where nobody worries too much about packets that never reach their destination.

In addition, mainframe networking, in general, has got significantly more complex and convoluted over the last few years. Today, when dealing with mainframe networks, one has to contend with multiple IP stacks per LPAR, Virtual IP Addresses (VIPA), dynamic VIPA takeovers across Sysplex, gigabit speed OSA interfaces, multi-host environments, and intrusion detection. That is just on the mainframe side. The network *per se*, will, in addition, consist of routers, switches, firewalls, tn-servers, and possibly even Linux LPARs.

But this is only part of the problem when it comes to mainframe IP monitoring and management. The complexity is exacerbated by the absence of standard schemes or techniques, at least at present, to facilitate mainframe-centric, IP network monitoring and management. This statement may come as a surprise to those who believe that the Simple Network Management Protocol (SNMP) is the standard, accepted, and preferred means for any and all types of IP network management and monitoring. SNMP, however, may not necessarily be the optimum means for monitoring mainframe-resident IP resources or IP traffic flowing within a mainframe. The issue with SNMP has to do entirely with overhead and thoroughness.

Keeping overheads down to a minimum is a crucial, and often overlooked, factor when selecting mainframe network management tools. The concern here, in reality, is not resource (eg CPU cycles, memory, I/O bandwidth) consumption. Today's mainframes, particularly with their cutting-edge resource virtualization technology and 'capacity-on-demand' features,

are not known for being short on computing power. Nonetheless, one does not want network management to get in the way of the production work that it is supposed to be monitoring, nor for it to be so wrapped up in what it is doing that it cannot deliver real-time visibility. A good mainframe monitor should be nimble, light, and sparing when it comes to CPU utilization. Ideally CPU utilization should be around the 1% mark. Relying on SNMP alone for mainframe-related IP statistics and status, as is the case with many well-known mainframe monitors, could preclude this CPU utilization goal from being met.

HITTING THE STACK WHEN YOU CAN

SNMP, along with the necessary Management Information Base (MIB) 'data structures', though fully supported by today's primary mainframe OSs, is not a standard default capability that is automatically activated during an IPL. Instead, in order to be able to use SNMP in the context of a mainframe, one first has to configure and activate the so-called 'OSNMP daemon' – for each mainframe IP stack that needs to be monitored and managed. Once the daemons have been configured and activated, the daemon for each stack has to be repeatedly polled (given that SNMP is a request/response protocol), via UDP packets, to obtain the necessary data from the MIBs.

This daemon-based approach, following the precepts of SNMP, works and does what it is supposed to do. Hence the popularity of this approach in many of today's mainframe IP monitors. But using SNMP to obtain mainframe-specific IP data may not be optimum. In some instances, the SNMP MIB queries, depending on the parameters of the request, can result in large amounts of superfluous data, such as lengthy connection tables being sent back from the MIB. Plus, the request/response nature of SNMP also dictates the need for frequent polling of the MIB to obtain up-to-date statistics. These frequent polls, and the data they generate, particularly if some of this data is redundant, can result in the consumption of considerable mainframe resources. Fortunately there is an elegant workaround.

Everything IP-related on a mainframe revolves around a mainframe IP stack. The IP stack is akin to what ACF/VTAM (and its embedded SSCP) used to be in the SNA era. All TCP/IP-related data maintained by a MIB must come, to begin with, from, or via, a mainframe IP stack. The z/OS stack, for example, maintains 43 separate performance statistics related to just TCP/IP traffic. In general, all the same TCP/IP monitoring data that SNMP can obtain from the standard mainframe TCP/IP and IBM Enterprise MIBs can be accessed directly from the stack – using IBM provided cross-memory APIs. Getting the relevant management data from the stack, rather than a MIB, cuts down on mainframe overhead. Typically you can get the same information (and in some cases even more) using less bandwidth.

But this is not to say that SNMP is *passé*. *Au contraire*. Direct, cross-memory, stack access is only possible with mainframe-resident stacks. SNMP-based MIB access will still be required to get data from external IP hosts such as routers, switches, firewalls, and even Linux partitions. Some mainframe monitors, in addition, opt for packet tracing, or z/OS NetStat command-based 'screen scraping' to obtain TCP/IP data to facilitate mainframe-centric IP monitoring. Ideally packet-tracing should be a diagnostic adjunct to stack access and SNMP rather than being the main technique for TCP/IP monitoring. Relying on packet tracing alone can lead to 'blind spots' – in particular, lack of timely visibility to 'hung' resources. This is because packet tracing cannot detect, or see, packets that are getting dropped because a resource is in distress. A future article will address this in more detail.

BOTTOM LINE

IP is now the sole basis for mainframe networking, with SNA transactions being conveyed across IP fabrics. Despite the success of IP, mainframe IP monitoring, however, is not as slick and sophisticated as some would like, particularly in comparison with what they were used to in the SNA world. Fortunately this is beginning to change, and change fast. Rather than relying

primarily on SNMP, packet-tracing, or NetStat 'screen scraping', mainframe tools are now beginning to synthesize multiple techniques to deliver more incisive and responsive visibility.

Anura Gurugé
Strategic Consultant (USA)

William Data Systems has announced Version 3.1 of IMPLEX, its real-time TCP/IP monitor for z/OS. The product provides monitoring and alerting capabilities for all IP traffic on z/OS.

IMPLEX monitors entire Sysplex configurations as well as single IP stacks and multiple IP stacks across multiple LPARs and CPUs. IMPLEX can reveal traffic whose existence has not been suspected and is able to track activity levels, issue alerts, and provide IP stack activity information.

Among new features are full support for Sysplex environments and hierarchical views from Sysplex down to individual LPARS. New granular alerts avoid message flooding. Problem determination can be speeded up by the use of user look-up facilities (IP address, LU name, user name, application). There is also a new TN3270 Response Time display providing additional information for more accurate TN3270 response time reporting.

For further information contact:
BMC Software, 2101 City West Blvd, Houston, TX 77042-2827, USA.
Tel: (713) 918 8800.
URL: http://www.willdata.com/v2/products/implex.htm.

* * *

SDS has announced Version 3 of Vital Signs VisionNet IP Monitor (VIP), its TCP/IP performance management software. The product monitors the health, configuration, and traffic levels on TCP/IP networks running on z/OS or OS/390, and produces real-time reports on OSA, FTP, telnet, and remote hosts.

It provides summary and history panels, which allow navigation to details for every single connection. It can track both current and past file transfers and telnet sessions. TCP/IP applications can be ranked according to the amount of data they're moving or the number of users.

VIP helps users to identify nodes that are no longer active, and it can monitor the path, path length, round-trip time, response time, and number of connections between a z/OS system and any remote host.

For further information contact:
Software Diversified Services, 5155 East River Road, Suite 411, Minneapolis, MN 55421-1025, USA.
Tel: (763) 571 9000.
URL: http://www.sdsusa.com/vip.

* * *

Zephyr has announced that PASSPORT terminal emulation software can now be used to provide a single sign-on implementation for IBM and Unix host systems when used with the v-GO SSO product from Passlogix.

PASSPORT terminal emulation provides access to IBM zSeries (3270), IBM iSeries (5250), and Unix (Telnet) host systems. The v-GO SSO single sign-on software product from Passlogix provides single sign-on to these host systems as well as client/ server, Web-based, desktop-based, and other types of applications as well. The benefits of using single sign-on with a 3270, 5250, or VT100/VT220 terminal emulator program include increased productivity and greater security.

For further information contact:
Zephyr, 8 E Greenway Plaza, Suite 1414, Houston, TX 77046, USA.
Tel: (713) 623 0089.
URL: http://www.zephyrcorp.com/3270-telnet-emulator.htm.