59 TCP/SNA update

September 2005

In this issue

A xephon PUBLICATION

Published by

Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

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

Editor

Trevor Eddolls
E-mail: trevore@xephon.com

Publisher

Colin Smith
E-mail: info@xephon.com

Subscriptions and back-issues

A year's subscription to *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.

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.

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.

Printed in England.

REXEC versus FTP in change management

At my site we have the development and production environments physically separated – they are located more than 180km apart.

For this reason we had to integrate into our change management cycle (for moving applications to the production environment) an FTP step. This solution generated a performance problem, particularly when FTPing single load modules.

So we decided to experiment with another method of moving programs to production – we decided to unload load modules to a sequential dataset, transfer that by FTP, and then invoke on the remote site a CLIST reversing the load with IEBCOPY to the final library.

This solution, as I'll show below, not only gives great performance enhancements, but it also allows the sending task to have a return code reflecting exactly the status of the move to production. This information is important to the programmer requesting the move.

The remote execution of the CLIST is possible thanks to REXEC (Remote EXECution), the TSO Remote Execution Server that enables TSO commands to be submitted from a remote host and executed on z/OS. The configuration of this server is explained in the *Communication Server IP Configuration Reference* manual, and is really very easy.

The server, when it receives a request, submits a TSO batch job using another proc whose sysin is the remote command.

First of all you need to open both the firewalls of the remote sites on two ports (512 and 514 are default), then the same ports must be specified in the TCP/IP of both mainframes.

Then you need to copy to a system proclib the started task RXSERVE:

```
//RXSERVE PROC MODULE='RSHD',
//          EXIT=,
//          TSOPROC=RXPROC,
//          MSGCLASS=X,
//          TSCLASS=X,
//          MAXCONN=512,
//          PREFIX=XXXX,
//          PURGE=N,
//          TRACE=
//*         TRACE=LOG
//RXSERVE  EXEC PGM=&MODULE,PARM=('EX=&EXIT,TSO=&TSOPROC',
//              'MSG=&MSGCLASS,TSC=&TSCLASS',
//              'MAX=&MAXCONN,PRE=&PREFIX,TR=&TRACE',
//              'PUR=&PURGE'),
//             TIME=1440
//STEPLIB  DD DISP=SHR,DSN=TCPIP.SEZATCP
//SYSPRINT DD SYSOUT=*
//SYSTCPD  DD DSN=SYS1.TCPPARMS(TCPDATA),DISP=SHR
```

Be careful in particular with the following parameters:

- *TSOPROC* is the name of the procedure that executes the remote command via TSO batch.
- *TSCLASS* is the sysout class of the submitted job.
- *MSGCLASS* is the msgclass of the submitted job.
- *PREFIX* is the four-character prefix of the submitted job.

RXPROC is the procedure used in the job submitted by RXSERVE and it also needs to be located in a system proclib. It looks like this:

```
//RXPROC   PROC
//IKJACCNT EXEC PGM=IKJEFT01,DYNAMNBR=500,REGION=4M,TIME=1440
//SYSPROC  DD DISP=SHR,DSN=MY.ISPCLIB
//SYSHELP  DD DISP=SHR,DSN=SYS1.HELP
//         DD DISP=SHR,DSN=ISF.SISFHELP
//         DD DISP=SHR,DSN=SYS1.SBDTHELP
//         DD DISP=SHR,DSN=SYS1.HELPENP
//         DD DISP=SHR,DSN=ISP.SISPHELP
//SYSLBC   DD DISP=SHR,DSN=SYS1.BRODCAST
//SYSPRINT DD SYSOUT=*
//SYSUT3   DD UNIT=3390,SPACE=(CYL,(90,50))
//SYSUT4   DD UNIT=3390,SPACE=(CYL,(90,50))
```

The sysproc MY.ISPCLIB must include the remote commands requested to RXSERVE.

Obviously, the userid assigned to RXSERVE needs to be the OMVS segment.

Normally, RXSERVE transmits to the sending user all the messages issued during the execution of the command. To make them available on the sending site in a formatted way and in a separated sysout, and to make them more readable, I wrote a small REXX called REXIT, which you need to install in a CLIST library at the sending site:

```
/* rexx REXIT*/
"execio * diskr inprint (stem inprint. finis)"
oldj = 1
row  = ''
do i = 1 to inprint.Ø
   l = length(inprint.i)
   do j = oldj to 133
      if j > l then iterate i
      row = row||substr(inprint.i,j,1)
      end
   say row
   if index(row,'IEB147I') > Ø then do
      parse var row 'IEB147I END OF JOB -' ccode .
      exit(ccode)
      end
   oldj = 1
   row  = ''
   end
exit(12)
```

Here is the job that passes the load module to production:

```
//PASSPROD JOB (XC99SØ56CBØ1Ø),MSGCLASS=5,CLASS=4
//*************************************************************
//* first of all delete the sequential used for transmission
//*************************************************************
//S1 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
 DELETE TEMP.SEQ
 SET MAXCC = Ø
//*************************************************************
//* then unload to the sequential ds the module to transfer
//* to the remote site
//*************************************************************
//CMDLIB   EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//SYSUT3 DD SPACE=(CYL,(99,8)),UNIT=339Ø
//SYSUT4 DD SPACE=(CYL,(99,8)),UNIT=339Ø
```

```
//I1 DD DSN=DEVL.LOADLIB,DISP=SHR
//O1 DD DSN=TEMP.SEQ,DISP=(,CATLG,DELETE),UNIT=339Ø,
//    SPACE=(CYL,(3Ø,1Ø))
//SYSIN DD *
 COPY INDD=((I1,R)),OUTDD=O1
 S M=loadpgm
//*************************************************************
//* now transfer by FTP to the production site the sequential ds
//*************************************************************
//FTPEB    EXEC PGM=FTP,REGION=2M,PARM='(EXIT TIMEOUT 6ØØ'
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//INPUT    DD *
remote.ip.address
userid
password
mode b
type e
SENDSITE
SITE LRECL=X RECFM=VS BLKSIZE=32760 CY PRI=1Ø
PUT 'TEMP.SEQ'   'TEMP.SEQ'
QUIT
/*
//*************************************************************
//* now call RXSERVE on the remote site and ask it to
//* execute the CLIST REMCOPY with the names of the input
//* and output dataset
//*************************************************************
//REXEC   EXEC PGM=IKJEFTØ1
//SYSUT3   DD UNIT=DISK,SPACE=(CYL,(5Ø))
//SYSUT4   DD UNIT=DISK,SPACE=(CYL,(5Ø))
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD DSN=&&TEMP,DISP=(,PASS),UNIT=SYSDA,
//         DCB=(LRECL=133,RECFM=FB,BLKSIZE=Ø),SPACE=(CYL,1)
//SYSTSIN  DD *
REXEC -l userid -p password  remote.address.ip %REMCOPY -
'TEMP.SEQ' 'PROD.LOADlib'
//*************************************************************
//* and lastly format the answer returned from the CLIST
//* executed remotely
//*************************************************************
//STEPØ5   EXEC PGM=IKJEFT1B
//SYSEXEC  DD DSN=DEVL.ISPCLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//INPRINT  DD DSN=&&TEMP,DISP=(OLD,DELETE)
//SYSOUT   DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*,DCB=LRECL=133
//SYSTSIN  DD *
 REXIT
```

The step *REXEC* calls the remote RXSERVE and asks it to

execute the CLIST REMCOPY, passing two parameters – the name of the sequential dataset previously transferred by FTP and the remote production loadlib library.

The CLIST REMCOPY is located at the production site in MY.ISPCLIB, just as referred to by RXPROC.

Here is REMCOPY:

```
PROC 2 FILEIN PDSOUT
CONTROL CONLIST SYMLIST LIST MSG
ALLOC FILE(I1)     DSN(&FILEIN) SHR
ALLOC FILE(O1)     DSN(&PDSOUT) SHR
ALLOC FI(SYSIN) DSN('MY.ISPCLIB(RXSYSIN)') SHR REUSE
CALL 'SYS1.LINKLIB(IEBCOPY)'  'WORK=32M'
FREE FI(I1)
FREE FI(O1)
FREE FI(SYSIN)
EXIT CODE(&LASTCC)
```

where *RXSYSIN* is:

```
COPY INDD=((I1,R)),OUTDD=O1
```

Here is an example of the job remotely submitted by RXSERVE:

```
//XXXX27Ø  JOB userid,
// USER=userid,
// PASSWORD=,
// MSGCLASS=X
//TSO EXEC RXPROC
//DEFAULT  OUTPUT DEFAULT=YES,CONTROL=PROGRAM,WRITER=RSHD
//SYSTSPRT DD SYSOUT=(X,RSHD),HOLD=YES
//SYSPRINT DD RECFM=VBA,SYSOUT=(*,RSHD),HOLD=YES
//SYSTERM  DD RECFM=VBA,SYSOUT=(*,RSHD),HOLD=YES
//SYSTSIN  DD *
%REMCOPY 'TEMP.SEQ' 'PROD.LOADLIB'
/*
```

Finally, just a few numbers:

- A simple and direct FTP transfer of a load module on average takes about three minutes.
- The entire job with REXEC takes about 26 seconds, where REXEC itself uses almost 19 seconds.

Well, happy remote execution!

Maria Elena Campidoglio
Systems Programmer (Italy)

Automated syntax checking for VPS member lists

We have been running VPS in our data centre for many years. When we first purchased the software, our printer population was almost exclusively SNA based. In more recent years, there has been a steady migration from SNA-based to TCP/IP-based printers. VPS is a well-behaved application that has handled the changing print environment very well. Regardless of the protocols involved, our support centre personnel build the printer definitions, test the connectivity, and implement the printers in our VPS network. This is generally a simple and error-free process. We have encountered two problems that are the direct result of human error. They are:

- A missing or extraneous comma in the VPS member list.
- A failure to update the MAXPRTS parameter when adding printers.

The first problem deals with the syntax of the VPS initialization member list. In order for a printer to be activated in VPS, it must be specified in a VPS member list. VPS supports both inclusive and exclusive member lists for printer activation. The inclusive list specifies the printers that VPS should activate; the exclusive list specifies the printers that VPS should *not* activate. We have always used the inclusive member list, although the syntax rules are the same for both types of list. A printer name must be specified on a line starting between columns 2 and 16. The first column of the line must have a blank. If the first column of a line contains an asterisk, the line is considered to be a comment. All printer names except the last must be followed by a comma. The last printer specified must be followed by one space and no comma. Comments can also be placed on lines that have printers specified as long as there is one space after the control information. Although you can code multiple printers on a single line, it has always been our convention to code only one printer on a line. A brief example follows:

```
* This entire line is a comment
 PRINTER1,      comments on the rest of this line
 PRINTER2,      comments on the rest of this line
 LASTPRTR       comments on the rest of this line
```

Although these rules are relatively simple to follow, we have had instances of someone forgetting a comma or coding the printer name in column 1. In this event, VPS abends at initialization with a U0005 abend code. In our environment, this is usually found over the weekend, at system IPL time. This results in the weekend operations personnel 'beeping' the on-call support centre to resolve the situation.

The second problem that we have encountered deals with the maximum number of printers that can be active in the VPS started task. The maximum number of printers that VPS can activate is controlled by the MAXPRTS initialization parameter. This parameter is coded as follows:

```
MAXPRTS=(xxxx,yyyy)
```

The first positional parameter, *xxxx*, indicates the maximum number of printers that VPS can activate. An attempt to activate more than *xxxx* printers results in an error message and the failure of the activate command. The second positional parameter, *yyyy*, indicates the maximum number of printers that VPS expects to be printing at any one time. This second value can be increased dynamically via VMCF (VPS Monitor and Control Facility). Once the first value (*xxxx*) has been reached, the only way to activate more printers is either to inactivate some that are already active, or to increase the MAXPRTS initialization parameter and recycle the VPS started tasks.

Although these problems do not occur frequently, their impact is rather severe. The balance of this article is intended to provide a tool to assist in avoiding these types of problem. We have created two REXX EXECs that check the VPS member list for syntax errors and additionally compare the number of printers specified with the current value of MAXPRTS.

The first REXX EXEC was written for use as a NetView

command. It was intended as an interactive tool and it exploits the NetView PIPE command. Originally, I envisioned that the support centre individuals who made the VPS changes would subsequently run this EXEC to determine whether there were any syntax errors or whether a MAXPRTS change was necessary. If no input parameters were provided, the REXX EXEC would determine the SMFID of the system where it was invoked and, based on our conventions, it would determine the name of the VPS initialization member list. For example, if the EXEC was invoked on our system 'F' (the SMFID would be SYSF), then the VPS member list name would be VPSMLSTF. The EXEC would count the number of printers and it would determine the current value of the MAXPRTS parameter. These values are provided in messages that are written to the NetView console. An example follows:

```
* CNM02    CHKVPSML
C CNM02    CHKVPSML Dataset is SYS2.VPS.CNTL(VPSMLSTC)
C CNM02    CHKVPSML Processed 100 printers in member list VPSMLSTC
C CNM02    CHKVPSML Processed 200 printers in member list VPSMLSTC
C CNM02    CHKVPSML Processed 300 printers in member list VPSMLSTC
C CNM02    CHKVPSML Processed 400 printers in member list VPSMLSTC
C CNM02    CHKVPSML Processed 500 printers in member list VPSMLSTC
C CNM02    CHKVPSML Processed 600 printers in member list VPSMLSTC
C CNM02    CHKVPSML Processed 700 printers in member list VPSMLSTC
C CNM02    CHKVPSML Processed 800 printers in member list VPSMLSTC
C CNM02    CHKVPSML Processed 900 printers in member list VPSMLSTC
C CNM02    CHKVPSML Processed 1000 printers in member list VPSMLSTC
C CNM02    CHKVPSML Processed 1100 printers in member list VPSMLSTC
C CNM02    CHKVPSML Processed 1200 printers in member list VPSMLSTC
C CNM02    CHKVPSML Processed 1300 printers in member list VPSMLSTC
C CNM02
C CNM02    CHKVPSML has ended - 1347 VPS Printers in VPSMLSTC
C CNM02    CHKVPSML has ended - Max printers for VPS on C is 2100
C CNM02    CHKVPSML has ended - 0 Syntax errors Found in VPSMLSTC
```

In the above example, no syntax errors were found in the VPS member list, VPSMLSTC. Additionally, 1347 printers are specified in the member list and the current value of MAXPRTS is 2100. The EXEC determines the MAXPRTS value by checking a NetView common global variable. This variable is set via NetView automation after the VPS task has successfully initialized. We use a NetView CLIST language command named VPSDSPSY, which is triggered from the VPS000N

message. The VPSDSPSY CLIST issues a VPS DISPLAY SYSTEM command and parses the output. The VPS971R message provides the current value for the MAXPRTS parameter. The CLIST captures this value and stores it in a NetView common global variable. The global variable name is built by concatenating the VPS started task name with the characters 'MXP'. The code for the VPSDSPSY command appears later in this article.

If the number of printers specified exceeds the value of MAXPRTS, then the EXEC issues a warning message. An example of this follows. This time there is a syntax error in the member list and the number of printers specified exceeds the MAXPRTS parameter:

```
CNM17    CHKVPSML VPSMLSTX
CNM17    CHKVPSML Dataset is SYS2.VPS.CNTL(VPSMLSTX)
CNM17    CHKVPSML Processed 100 printers in member list VPSMLSTX
CNM17    CHKVPSML Processed 200 printers in member list VPSMLSTX
CNM17    CHKVPSML Processed 300 printers in member list VPSMLSTX
CNM17    CHKVPSML Processed 400 printers in member list VPSMLSTX
CNM17    CHKVPSML Processed 500 printers in member list VPSMLSTX
CNM17
CNM17  CHKVPSML found a comma after KWD= TESTPR98, of VPSMLSTX prt = 549
CNM17     ------------------------------------------------------
CNM17
CNM17
CNM17    CHKVPSML has ended - 549 VPS Printers in VPSMLSTX
CNM17    CHKVPSML has ended - Max printers for VPS on X is 500
CNM17    CHKVPSML has ended - 1 Syntax errors Found in VPSMLSTX
CNM17
CNM17    WARNING VPSMLSTX has more printers defined than VPS can start!!
CNM17    WARNING VPSMLSTX has more printers defined than VPS can start!!
```

In the above example, a syntax error was found on the last line of the VPS member list. The printer coded on the last line was followed by a comma. Please note that the error message indicated the printer name (KWD=TESTPR98) where the error was encountered. Also notice that the error message indicates that the error was found on prt=549. This does not mean that the error occurred on line 549 of the member list; rather, it indicates that the error occurred on the 549th printer.

The CHKVPSML REXX EXEC follows:

```
/* REXX */
/*********************************************************************/
/*    MEMBER-NAME: CHKVPSML                                          */
/*    REXX EXEC to syntax check the appropriate VPS member list.     */
/*********************************************************************/
parse arg mbrl
trace off
sc = substr(cursys(),4,1)
jobn = vps
if mbrl = '' then mbrl = 'VPSMLST'sc
if mbrl = 'VPSMLSTI' then jobn = 'VPSI'
/*  Customizable variables */
  if length(mbrl) ¬= 8 then
    do
      say = 'CHKVPSML invalid member list specified 'mbrl
      exit
    end
  da = Ø
  ic = 1ØØ
  sa      = substr(applid(),4,2)
  dataset = 'SYS2.VPS.CNTL'
  dataset = dataset||'('||mbrl||')'
  say 'CHKVPSML Dataset is 'dataset
/*  Main Processing  */
  'PIPE QSAM (DSN)'dataset'|',              /* Display all DESTIDs  */
     'NLOCATE 1.1 /*/ |',                   /* Remove the Comments  */
     'CHOP 11 |',                           /* Remove the VTAMPUs   */
     'STEM VPSLIST.'
  do i = 1 to (VPSLIST.Ø - 1)
/*   parse var VPSLIST.i prtname comma space */
       if substr(vpslist.i,1Ø,1) ¬= ',' then
         call run_error1 vpslist.i
       if substr(vpslist.i,11,1) ¬= ' ' then
         call run_error2 vpslist.i
       if substr(vpslist.i,1,1) ¬= ' ' then
         call run_error3 vpslist.i
       if substr(vpslist.i,2,1) = '*' then
         call run_error3 vpslist.i
     if i = ic then
       do
         say 'CHKVPSML Processed 'i' printers in member list 'mbrl
         ic = ic + 1ØØ
       end
  end
  if i = vpslist.Ø then
  do
   if substr(vpslist.i,1Ø,1) ¬= ' ' then
     call run_error4 vpslist.i
   if substr(vpslist.i,1,1) ¬= ' ' then
     call run_error3 vpslist.i
```

```
   if substr(vpslist.i,2,1) = '*' then
     call run_error3 vpslist.i
  end
   if jobn = 'VPS' then
     do
       'globalv getc vpsmxp'
       maxprts =  vpsmxp
     end
   if jobn = 'VPSI' then
     do
       'globalv getc vpsimxp'
        maxprts =  vpsimxp
     end
  say '                    '
  say 'CHKVPSML has ended - 'VPSLIST.Ø' VPS Printers in 'mbrl
  say 'CHKVPSML has ended - Max printers for VPS on 'sc' is 'maxprts
  say 'CHKVPSML has ended - 'da' Syntax errors Found in 'mbrl
/*if vpslist.Ø > vpsmxp then   */
  if vpslist.Ø > maxprts then
  do
  say '                    '
  say 'WARNING 'mbrl' has more printers defined than VPS can start!!'
  say 'WARNING 'mbrl' has more printers defined than VPS can start!!'
  exit -1
  end
exit
run_error1: procedure expose mbrl i da
  arg prtr
  say ' '
  say 'CHKVPSML missing comma after KWD='prtr' in 'mbrl' prt = 'i
   say '-------------------------------------------------------'
  say ' '
  da = da + 1
  'delay Ø1'
return
run_error2: procedure expose mbrl i da
  arg prtr
  say ' '
  say 'CHKVPSML 'prtr' must be follow by a '', '''mbrl' prt = 'i
   say '-------------------------------------------------------'
  say ' '
  da = da + 1
  'delay Ø1'
return
run_error3: procedure expose mbrl i da
  arg prtr
  say ' '
  say 'CHKVPSML found a syntax error,KWD='prtr' in 'mbrl' prt = 'i
   say '-------------------------------------------------------'
  say ' '
```

```
  da = da + 1
  'delay Ø1'
return
run_error4: procedure expose mbrl i da
  arg prtr
  say ' '
  say 'CHKVPSML found a comma after KWD='prtr' of 'mbrl' prt = 'i
  say '-------------------------------------------------------'
  say ' '
  da = da + 1
  'delay Ø1'
return
```

This EXEC is useful in avoiding syntax errors in the VPS member lists and virtually eliminates the number of VPS ABENDs occurring over the weekend. The only downside is that, since the EXEC is designed as an interactive tool, someone has to remember to use it before the weekend IPLs. Occasionally, this is overlooked. This lead us to attempt a more automated solution.

The next iteration of this EXEC is designed to run in batch mode. The intention is to build a batch job that could be executed every Friday. If the new EXEC determines that there is a syntax error, or if the number of printers specified in a particular member list exceeds the value of MAXPRTS, the EXEC returns a non-zero return code and the batch job ABENDs. The ABEND is tracked by our production services group and appropriate personnel are notified. If the job runs successfully, no further action is necessary. This also solved the problem of human intervention: no-one would have to remember to run the EXEC. This second batch version of the EXEC is written in TSO REXX and is unable to exploit the NetView PIPE command. In order to emulate the functionality of the PIPE QSAM command, the EXECIO command is used. We named the new version CHKVPSMX. The batch version of the EXEC follows:

```
/* REXX */
/*********************************************************************/
/*    MEMBER-NAME: CHKVPSMX     (BATCH VERSION)                      */
/*    REXX EXEC to syntax check the appropriate VPS member list.     */
/*********************************************************************/
signal on error
```

```
parse arg mbrl
trace off
CVT    = STORAGE(10,4)
DCVT   = C2D(CVT)
SYSTEM = DCVT  + 340
CLEVEL = D2X(SYSTEM)
smfid  = STORAGE(CLEVEL,4)        /* SYSTEM       */
sc = substr(smfid,4,1)
/* say 'userid = 'userid()' SMFID = 'smfid' sc = 'sc */
jobn = vps
if mbrl = '' then mbrl = 'VPSMLST'sc
if mbrl ¬= '' then
  do
       s8 = substr(mbrl,8,1)
       sc = s8
       strl = 'VPSTART's8
  end
if mbrl = 'VPSMLSTI' then
  do
       jobn = 'VPSI'
       strl = 'VPSTRTIG'
  end
if mbrl = 'VPSLSTC2' then
  do
       strl = 'VPSTRTC2'
  end
/*  Customizable variables */
  if length(mbrl) ¬= 8 then
    do
      say = 'CHKVPSMX invalid member list specified 'mbrl
      exit 100
    end
  da = 0
  ic = 100
  erc = 0
  com = 0
  j = 1
/*sa      = substr(applid(),4,2) */
  vpscntl = 'SYS2.VPS.CNTL'
  dataset = vpscntl||'('||mbrl||')'
  say 'CHKVPSMX Dataset is 'dataset
/*  Main Processing  */
"alloc ddname(VPSMLIST) DATASET('"dataset"') shr"
"EXECIO * DISKR VPSMLIST (FINIS STEM VVVLIST."
/*Strip out all of the comments*/
  do k = 1 to VVVLIST.0
     if substr(vvvlist.k,1,1) = '*' then
        com = com + 1
     else
        do
```

```
           vpslist.j = vvvlist.k
           j = j + 1
         end
  end
 vpslist.Ø = vvvlist.Ø - com
  do i = 1 to (VPSLIST.Ø - 1)
/*   parse var VPSLIST.i prtname comma space */
       if substr(vpslist.i,1Ø,1) ¬= ',' then
         call run_error1 vpslist.i
       if substr(vpslist.i,11,1) ¬= ' ' then
         call run_error2 vpslist.i
       if substr(vpslist.i,1,1) ¬= ' ' then
         call run_error3 vpslist.i
       if substr(vpslist.i,2,1) = '*' then
         call run_error3 vpslist.i
     if i = ic then
       do
         say 'CHKVPSMX Processed 'i' printers in member list 'mbrl
         ic = ic + 1ØØ
       end
  end
/*  Last Line processing */
  if i = vpslist.Ø then
  do
   if substr(vpslist.i,1Ø,1) ¬= ' ' then
     call run_error4 vpslist.i
   if substr(vpslist.i,1,1) ¬= ' ' then
     call run_error3 vpslist.i
   if substr(vpslist.i,2,1) = '*' then
     call run_error3 vpslist.i
  end
/*  Maxp Processing  */
if strl = '' then strl = 'VPSTART'sc
dataset = vpscntl||'('||strl||')'
  say 'CHKVPSMX Dataset is 'dataset
"alloc ddname(VPSSTART) DATASET('"dataset"') shr"
"EXECIO * DISKR VPSSTART (FINIS STEM VPSTART."
  do i = 1 to (VPSTART.Ø - 1)
       if substr(vpstart.i,1,1) = '*' then
         iterate
       if substr(vpstart.i,2,8) = 'MAXPRTS=' then
       do
      /* say 'Vpstart.i is 'vpstart.i  */
         parse var VPSTART.i vpskeyw comma filler
      /* say 'VPS Keyword is 'vpskeyw  */
         parse var vpskeyw keywd "=" value
      /* say 'Max prts value is 'value */
         parse var value "(" maxprts "," maxact ")"
         say 'MAXPRTS is 'maxprts
         say 'MAXACT  is 'maxact
```

```
      end
  end
  say 'CHKVPSMX has ended - 'vpslist.Ø' VPS Printers in 'mbrl
  say 'CHKVPSMX has ended - Max printers for 'jobn' on 'sc' is 'maxprts
  say 'CHKVPSMX has ended - 'da' Syntax errors Found in 'mbrl
  say 'CHKVPSMX ended - Return code 'erc
  say '                              '
/*if vpslist.Ø > vpsmxp then   */
  if vpslist.Ø > maxprts then
  do
  say '                    '
  say 'WARNING 'mbrl' has more printers defined than VPS can start!!'
  say 'WARNING 'mbrl' has more printers defined than VPS can start!!'
  exit -1
  end
exit erc
run_error1: procedure expose mbrl i da erc
  arg prtr
  say ' '
  say 'CHKVPSMX missing comma after KWD='prtr' in 'mbrl' prt = 'i
   say '-------------------------------------------------------'
  say ' '
  da = da + 1
  erc = 1
/*'delay Ø1' */
return erc
run_error2: procedure expose mbrl i da erc
  arg prtr
  say ' '
  say 'CHKVPSMX 'prtr' must be follow by a '', '''mbrl' prt = 'i
   say '-------------------------------------------------------'
  say ' '
  da = da + 1
  erc = 2
/*'delay Ø1' */
return erc
run_error3: procedure expose mbrl i da erc
  arg prtr
  say ' '
  say 'CHKVPSMX found a syntax error,KWD='prtr' in 'mbrl' prt = 'i
   say '-------------------------------------------------------'
  say ' '
  da = da + 1
  erc = 3
/*'delay Ø1'*/
return erc
run_error4: procedure expose mbrl i da erc
  arg prtr
  say ' '
  say 'CHKVPSMX found a comma after KWD='prtr' of 'mbrl' prt = 'i
```

```
   say '-------------------------------------------------------'
   say ' '
   da = da + 1
   erc = 4
/*'delay Ø1' */
return erc
```

A sample batch job step to execute the CHKVPSMX EXEC is:

```
//VPSMBØ1 EXEC PGM=IKJEFT1A
//STEPLIB   DD DSN=SCSP.LOAD,
//             DISP=SHR
//SYSEXEC   DD DSN=SYS2.REXX.EXEC,
//             DISP=SHR
//SYSPRINT  DD SYSOUT=*
//SYSTSPRT  DD SYSOUT=*
//SYSTSIN   DD *
 CHKVPSMX VPSMLSTF
/*
```

There are a few points to note:

- The PGM=IKJEFT1A is required to ensure that any non-zero condition code is passed back to the job's JCL.
- When the EXEC is invoked, we specify the VPS member list explicitly. In our environment, we have seven different VPS started tasks. Consequently, we code seven job steps, one for each member list. By coding separate job steps, we know which VPS member list has a problem. The job step that ends with a non-zero condition code indicates the member list that has the problem. If necessary, you could code them all in a single job step.

In addition to the obvious, there are a few subtle differences between the two EXECs. The CHKVPSML EXEC (NetView version) uses a real-time display (VPSDSPSY command) to determine the MAXPRTS value. The CHKVPSMX EXEC uses EXECIO to read the MAXPRTS value directly from the VPS system initialization member. Consequently, it is possible for these two versions to get different values for the MAXPRTS parameter for the same VPS started task, for example, if the MAXPRTS parameter has been changed but the corresponding VPS task has not been recycled. In this case, the NetView

common global variable would still reflect the MAXPRTS parameter value prior to the change. However, the CHKVPSMX EXEC, via the EXECIO command, would determine the new value for the MAXPRTS parameter. Also, since the CHKVPSML EXEC uses real-time commands, the MAXPRTS check works correctly only if the EXEC is executed on the same system that runs the VPS started task.

In our environment, we run a batch job every Friday to execute the CHKVPSMX EXEC for each production VPS member list. If the job ends with a non-zero return code, then appropriate support personnel are notified. However, some support centre individuals still use the CHKVPSML version from NetView. This generally occurs after we convert a new client or after a large number of VPS printers have been added. In our environment a combination of these two EXECs has eliminated VPS ABENDs caused by syntax errors. Additionally, we have avoided printer activation issues caused by an insufficient MAXPRTS parameter.

Finally, the code for the VPSDSPSY command and the NetView message table statements that invoke it follow:

```
&CONTROL ERR
&IF .&1 EQ .? &THEN &GOTO -HELP
&IF .&1 EQ .HELP &THEN &GOTO -HELP
&VP = &1
&IF .&VP EQ . &THEN &VP = 'VPS'
&IF .&1 EQ .VPSERVER &THEN &EXIT
&MYM = 'VPS978R'
&IF &1  EQ .VPSI &THEN &MYM = 'VPS979R'
&IF &1  EQ .VPSTCP &THEN &MYM = 'VPS979R'
&WAIT CONTWAIT SUPPRESS
&WAIT '&VP DIS,SYSTEM'            VPS*=-SYSCHK             +
                                  IEE342I=-IEE342I                    +
                                  *15=-ERRMSG                         +
                                  *ERROR=-ERRMSG *ENDWAIT=-PROCESS2
-SYSCHK
&IF &MSGID = &MYM &THEN GO
&IF &MSGID ¬= VPS971R &THEN &WAIT CONTINUE
PARSEL2R MSGSTR 1 2 /(/ MP X/61/ CP X/61/ PM /)/ 3
&WRITE TASK=&VP - &MSGSTR
&WAIT CONTINUE
-PROCESS2
&WRITE &VP,LOG,VPSDSPSY &CP &VP Printers to check!!
```

```
&MAXPRT = &CONCAT &VP 'MXP'
&CGLOBAL &MAXPRT
UPD &MAXPRT &MP
*WRITE MAX PRINTER VARIABLE is &MAXPRT
*WRITE MAX PRINTER VALUE    is &&MAXPRT
&EXIT
-ERRMSG
&WRITE VPSDSPSY Timeout or error message for task &VP
&EXIT
-IEE342I
&WRITE VPSDSPSY received message IEE342I task &VP is busy!!
&EXIT


*
*    VPS000N VPS INITIALIZATION SUCCESSFUL MESSAGE
*
 IF (LABEL:VPS000N) MSGID='VPS000N' & JOBNAME=JOBN
      THEN EXEC( CMD('VPSDSPSY 'JOBN ) ROUTE(ONE AUTOVPS));
*
```

Anthony J Cieri 4th
Senior Network Consultant
SEI Investments (USA)

IPv6

IPv6 has been with us for over a decade now. The standard was initiated in 1991 because of the numbering issue. The recommendations for a new protocol emerged in 1994, but more than 10 years later the protocol is still not widely adopted. The standard is, for all intents and purposes, complete, most major vendors have products and/or services available to support it and it could be deployed by almost any organization today; however, it is evident that even now the take-up is not great. There are a number of reasons for this: primarily, most of today's networking demands are still easily supported by IPv4, therefore few enterprises can justify the deployment of IPv6. One of the principal reasons for the

development of IPv6 was the predicted shortage of IPv4 address spaces. However, the increasing use of NAT and increasingly rigorous delegation policies have reduced this problem. Without the prime mover for the introduction of IPv6 there is no longer a strong enough reason for most users to deploy and offset the higher costs associated with it. Certainly, we are not seeing the vendors pushing the technology to any great degree.

Eventually new networks and Internet-enabled devices will increase in demand for IP addresses, which will put further pressure on IPv4 and so lead to a changeover of versions, but it is still a long way off.

John Edwards
Network Administrator (UK)

The *Update* family

In addition to *TCP/SNA Update*, the Xephon family of *Update* publications now includes *AIX Update*, *CICS Update*, *DB2 Update*, *MQ Update*, *MVS Update*, and *RACF Update*. Details of all of these can be found on the Xephon Web site at www.xephon.com.

Remote batch command

PROGRAMS' TARGET

The two REXX programs, client and server, use the TCP/IP protocol and communicate through the port defined by the user.

The programs try to:

1 Obtain a report containing important information about the operating system (IODF, master catalog, number of CPUs online, real memory, etc) from z/OS servers.

2 Send TSO commands to the z/OS servers.

They can also be used for the management of security (RACF or Top Secret) to reset passwords and for user control of other systems. For example:

```
TSO LU userid
TSO ALU userid PASSWORD(pwd) RESUME)
```

They can be used for commands in the OMVS environment (eg TSO netstat, TSO ping), and for generic TSO commands (eg TSO listc lev(sys1), etc).

The server EXEC is installed on all the z/OS partitions from which you want system reports or to which you want to send

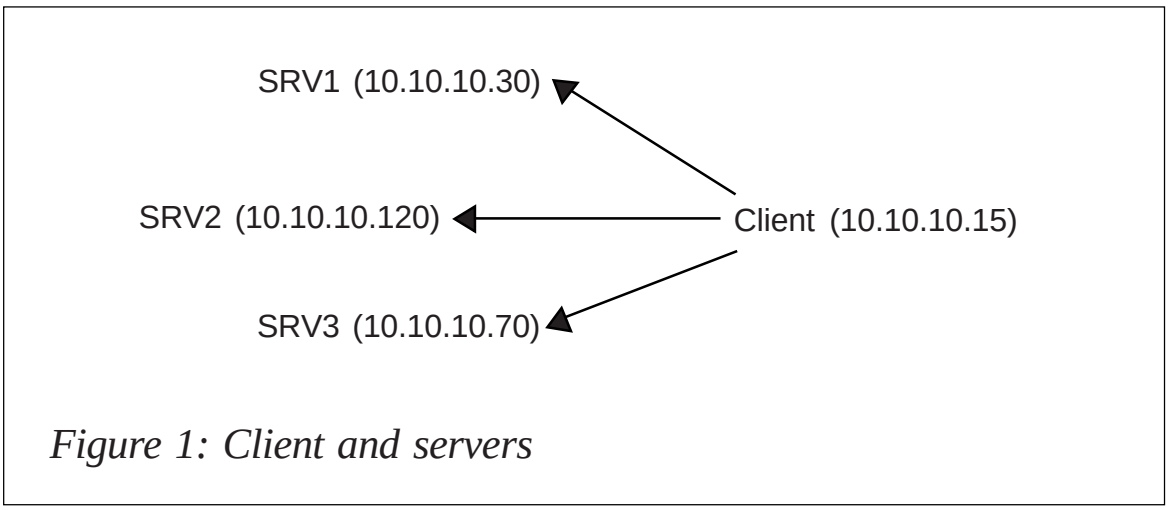

Figure 1: Client and servers

TSO commands. The client EXEC is installed on only one z/OS partition. Servers and client must be in the same network. There is one client, and many servers – see Figure 1.

The network address of this network configuration is 10.10.10.

The two programs allow there to be many servers and only one client from which the commands are executed.

SERVER PROGRAM SRVCMDS

```
/* REXX */
/*********************************************************************/
/* This is the basic server REXX.                                    */
/* It is called from a client and is passed a string.                */
/*                                                                   */
/* If the string starts with TSO, then all that follows will         */
/* be issued as a TSO command and the data trapped and returned to   */
/* the caller.                                                       */
/*                                                                   */
/* If the string SHUTDOWN is passed then the server will terminate   */
/*                                                                   */
/* If none of the above is passed then the string SERVER ERROR       */
/* followed by the string sent will be returned to the caller.       */
/*                                                                   */
/*  Syntax:                                                          */
/*    srvcmds port_number                                            */
/*                                                                   */
/*  Eg:                                                              */
/*   srvcmds 35000                (Server listen on port 35000)     */
/*********************************************************************/
/* */
SIGNAL ON syntax
lnecnt.=0
/* Initialization control information                        */
port = arg(1)
if port = ' ' then do
                    say " "
                    say " ************************************* "
                    say "  Invalid Syntax.The correct syntax is:"
                    say "  srvcmds port_number"
                    say "  Example : srvcmds 1946 "
                    say " ************************************* "
                    say " "
                    exit
                   end
/* */
/* Lpar name */
```

```
CVTECVT=D2X(C2D(STORAGE(10,4))+140)             /* Point to cvtsysad */
lparname=STRIP(STORAGE(D2X(C2D(STORAGE(CVTECVT,4))+344),8))
SAY 'SRVCMDS: initializing ....'
x= 'SOCKET'('Initialize','SRVCMDS')
IF WORD(x,1)¬='0' THEN DO
                        SAY 'ERROR while initializing server'
                        EXIT
                       END
/* We get the host IP address. This is done with a
   gethostid request. The first character returned is
   a success or failure indicator
   and in this case the second word is the IP address.          */
ipaddress='SOCKET'('GetHostId')
IF WORD(ipaddress,1)¬='0' THEN DO
                               SAY 'ERROR while getting hostid'
                               EXIT
                              END
ipaddress=WORD(ipaddress,2)
SAY 'SRVCMDS: initialised: ipaddress='ipaddress 'port='port
sock = 'SOCKET'('Socket')
IF WORD(sock,1)¬='0' THEN DO
                          SAY 'ERROR while getting socket'
                          EXIT
                         END
sock=WORD(sock,2)
x = 'SOCKET'('SetSockOpt',sock,'Sol_Socket','So_REUSEADDR','On')
x='SOCKET'('Bind',sock,'AF_INET' port ipaddress)
IF x¬=0 THEN DO
             SAY 'Error during AF_INET'
             EXIT
            END
/* Listen to the port. */
x='SOCKET'('Listen',sock)
IF x¬=0 THEN DO
             SAY 'Error during listen'
             EXIT
            END
/* Setting the io control mode with blocking. */
x='SOCKET'('Ioctl',sock,'FIONBIO','ON')
IF x¬=0 THEN DO
             SAY 'Error during set of io control mode'
             EXIT
            END
x='SOCKET'('Fcntl',sock,'F_SETFL','BLOCKING')
IF x¬=0 THEN DO
             SAY 'Error during set of io control mode'
             EXIT
            END
/* Wait for new connections and send lines. The array lnecnt will */
/* be used to keep track of data sent to each caller.               */
```

```
lnecnt. =     Ø
DO FOREVER
sellist='SOCKET'('SELECT','Write * Read * Exception')
PARSE UPPER VAR sellist . 'READ' rsock . 'WRITE' wsock . 'EXCEPTION' .
/* Now receive the information. If the socket id passed is the same */
/* as the one we are listening on, then we need to accept the       */
/* new connection.                                                  */
IF rsock¬='' THEN DO
   IF rsock=sock THEN DO
      x = 'SOCKET'('Accept',rsock)
      IF WORD(x,1) ¬='Ø' THEN DO
                                SAY 'Error adding another socket'
                                EXIT
                              END
                              ELSE rsock=WORD(x,2)
                        END
   x='SOCKET'('Recv',rsock)
   PARSE VAR x x . user string
   IF x¬='Ø' THEN DO
      SAY 'Connection lost'
      x='SOCKET'('Close',rsock)
      END
   ELSE DO
        stringuser.rsock=user
        stringword.rsock=string
        SAY 'User' user 'issued command' string 'at' TIME() DATE('E')
   END
END
/* Retrieve the command for this socket request and build the    */
/* information in the variable array msg.wsock.msgnum.            */
/* It is assumed that RESULT will contain the number of lines    */
/* to return to the caller upon return from the subroutine.      */
/* If it doesn't then 1 line to return is assumed.               */
/*                                                               */
/* Lines will be returned one at a time and the lnecnt for       */
/* the write socket will gradually drop to zero as data leaves   */
/* for the client.                                               */
IF wsoc¬='' THEN DO
   IF lnecnt.wsock=Ø THEN DO
      PARSE VAR stringword.wsock command parms
      SELECT
        WHEN command='SYSINFO' THEN call sysinfo
        WHEN command='TSO' THEN CALL tsocmds_process
        WHEN command='SHUTDOWN' THEN SIGNAL shutdown
        OTHERWISE CALL error
      END
      IF RESULT='' THEN lnecnt.wsock=1
      ELSE lnecnt.wsock=RESULT
   END
   msgnum=lnecnt.wsock
```

```
   msg=lparname msg.wsock.msgnum||'ØD'x
   x='SOCKET'('Send',wsock,msg)
   IF WORD(x,1)='Ø' THEN DO
      lnecnt.wsock = lnecnt.wsock - 1
      DROP msg.wsock.msgnum
      END
   IF WORD(x,1) ¬='Ø' THEN DO /* send failure - cleanup */
      lnecnt.wsock=Ø         /* indicate no lines      */
      DO x=1 TO msgnum
         DROP msg.wsock.x    /* release storage */
      END
      DROP stringword.wsock
      DROP stringuser.wsock
   END
   IF lnecnt.wsock=Ø THEN DO
      x='SOCKET'('Close',wsock)
      END
   END
END
/* Terminate the server and exit */
shutdown:
x='SOCKET'('Terminate')
SAY 'SRVCMDS: Terminated'
EXIT Ø
/****************** Subroutines ********************/
tsocmds_process:
ADDRESS TSO
CALL OUTTRAP('LINE.')
''parms
j=line.Ø
DO x=1 TO line.Ø
msg.wsock.j=line.x
j=j-1
DROP line.x
END
CALL OUTTRAP('OFF')
RETURN line.Ø
error:
msg.wsock.1='SERVER ERROR' stringword.wsock
RETURN 1
sysinfo:
Numeric digits 1Ø
yy=substr(date('J'),1,2)
ddd=substr(date('J'),3,3)
trat = copies('-',7Ø)
/******************
 ***   Pointer ****
 ******************/
CVT_PTR  = D2X(C2D(Storage(1Ø,4)))       /* point to CVT         */
SMCA_PTR = get_ptr(CVT_PTR,'C4')         /* point to SMCA        */
```

```
CVTECVT_PTR = get_ptr(CVT_PTR,'8C')      /* point to CVTECVT     */
ASMVT_PTR = get_ptr(CVT_ptr,'2CØ')       /* point to ASMVT       */
JESCT_PTR = get_ptr(CVT_ptr,'128')       /* point to JESCT       */
RESUCB_PTR = get_ptr(JESCT_PTR,'4')
CVTSYSAD_PTR = get_ptr(CVT_PTR,'3Ø')
CVTEXT2_PTR = get_ptr(CVT_PTR,'148')     /* point to CVTEXT2     */
CVTIXAVL_PTR = get_ptr(CVT_PTR,'7C')     /* point to IOCM        */
IOCIOVTP_PTR = get_ptr(CVTIXAVL_PTR,'DØ') /* pt to IOS Vect Tbl  */
CDA_PTR  = get_ptr(IOCIOVTP_ptr,'18')    /* point to CDA         */
CVTATCVT_PTR = get_ptr(CVTEXT2_PTR,'4Ø')
CVTATCVT_PTR = substr(CVTATCVT_PTR,3)
ISTATCVT_PTR = get_ptr(CVTATCVT_PTR,'Ø')
RCE_PTR = get_ptr(CVT_PTR,'49Ø')         /* point to RCE         */
CSD_PTR = get_ptr(CVT_PTR,'294')         /* point to CSD         */
PCCAVT_PTR = get_ptr(CVT_PTR,'2FC')      /* point to PCCA vect tb*/
CVTRAC_PTR = get_ptr(CVT_PTR,'3EØ')      /* point to RACF CVT    */
CVTDFA_PTR = get_ptr(CVT_PTR,'4CØ')      /* point to DFP ID table*/
CVTTVT_PTR = get_ptr(CVT_PTR,'9C')       /* point to TSO vect tbl*/
/*********************
 ***   Data_value ****
 *********************/
PLEXNM   = get_data(CVTECVT_PTR,'8',8)   /* point to SYSPLEX name*/
IPLVOL   = get_data(RESUCB_PTR,'1C',6)
PRODNAME = get_data(CVT_PTR,'-28',7)
FMIDNUM  = get_data(CVT_PTR,'-2Ø',7)
LPARNAME=STRIP(get_data(CVTECVT_PTR,'158',8))
IPLPARM = get_data(CVTECVT_PTR,'AØ',8)   /*  point to LOAD PARM  */
IPLPARM  = Strip(IPLPARM,T)              /* del trailing blanks  */
SEPPARM  = Substr(IPLPARM,1,4) Substr(IPLPARM,5,2),
           Substr(IPLPARM,7,1) Substr(IPLPARM,8,1)
SMFNAME  = get_data(SMCA_PTR,'1Ø',4)     /* point to SMF name    */
IPLTIME  = get_data(smca_ptr,'15Ø',4)    /* IPL Time - binary    */
IPLDATE  = get_data(smca_ptr,'154',4)    /* IPL Date - ØCYYDDDF  */
ATCVTLVL = get_data(ISTATCVT_PTR,'Ø',8)
JESNAME  = get_data(JESCT_PTR,'1C',4)    /* point to JESNAME     */
GRSNAME  = get_data(CVT_PTR,'154',8)     /* point to system name */
GRSNAME  = Strip(GRSNAME,T)              /* del trailing blanks  */
SMFNAME  = get_data(SMCA_PTR,'1Ø',4)     /* point to SMF name    */
RCVTID   = get_data(CVTRAC_PTR,'Ø',4)    /* point to RCVTID      */
                                         /* RCVT, ACF2, or RTSS  */
RACFVRM  = get_data(CVTRAC_PTR,'268',4)  /* RACF Ver/Rel/Mod     */
DFAPROD  = get_data(CVTDFA_PTR,'1Ø',1)   /* point to product byte*/
DFAPROD = C2D(DFAPROD)
STORSIZE = get_data(CVT_PTR,'358',4)     /* point to storage size*/
STORSIZE= C2D(STORSIZE)
STORSIZE = (STORSIZE/1Ø24)               /* Convert to megabytes */
MODEL    = get_data(CVT_PTR,'-6',2)      /* point to cpu model   */
MODEL =  C2X(MODEL)
NUMCPU   = get_data(CSD_PTR,'A',2)       /* point to # of CPUS   */
NUMCPU = C2X(NUMCPU)
```

```
call ipl_timeconv
call ipl_dateconv
call clpa_check
call ipl
call iodf
call master_catalog
title="System :" smfname "         Lparname :" lparname
title = center(title,72)
record = '0d25'x
record = record||trat||'0d25'x
record = record||title||'0d25'x
record = record||trat||'0d25'x
record = record||'0d25'x||'0d25'x
r1 = "Today is" date('w')  date('N') "(20"yy"."ddd")"
record = record||r1||'0d25'x
record = record||'0d25'x
r2 = "----- IPL -----"
record = record||r2||'0d25'x
r3 = "  The last Ipl was "ipldate" at "ipltime " "iplclpa
record = record||r3||'0d25'x
r4 = "  The system IPL address was "IPLADDR" volume:" IPLVOL
record = record||r4||'0d25'x
r5 =  '  The IPL LOAD PARM used was 'IPLPARM' ('SEPPARM')'
record = record||r5||'0d25'x
record = record||'0d25'x
r6 ="----- Sysplex -----"
record = record||r6||'0d25'x
r7 = "  The Sysplex name is :" PLEXNM
record = record||r7||'0d25'x
record = record||'0d25'x
r8 = "----- IODF -----"
record = record||r8||'0d25'x
r9 = '  The currently active IODF data set is 'IODF
record = record||r9||'0d25'x
r10 = '   Configuration ID =' CONFIGID ' EDT ID =' EDT
record = record||r10||'0d25'x
r11 = '   TOKEN: Processor  Date      Time      Description'
record = record||r11||'0d25'x
r12 = '          'IOPROC'   'IODATE'  'IOTIME'  'IODESC
record = record||r12||'0d25'x
record = record||'0d25'x
r13 = "----- Master catalog -----"
record = record||r13||'0d25'x
r14 = ' The Master Catalog is 'MCATDSN' on 'MCATVOL'.'
record = record||r14||'0d25'x
record = record||'0d25'x
r15 = "----- System Software -----"
record = record||r15||'0d25'x
call os_version
record = record||r150||'0d25'x
```

```
record = record||'0d25'x
call vtam_version
r16 = ' The VTAM Level is 'VTAMLEV'.'
record = record||r16||'0d25'x
record = record||'0d25'x
r17 = ' The primary job entry subsystem is 'JESNAME
record = record||r17||'0d25'x
record = record||'0d25'x
r18 = ' The GRS system id is 'GRSNAME
record = record||r18||'0d25'x
record = record||'0d25'x
r19 = ' The SMF system id is 'SMFNAME
record = record||r19||'0d25'x
record = record||'0d25'x
call racf
record = record||r190||'0d25'x
record = record||r191||'0d25'x
record = record||'0d25'x
call dfp
r20 = ' The' DFPRD 'level is' DFLEV'.'
record = record||r20||'0d25'x
record = record||'0d25'x
call tso
r21 = ' The TSO level is' TSOLEV
record = record||r21||'0d25'x
record = record||'0d25'x
r22 = "----- Real Storage Size -----"
record = record||r22||'0d25'x
r23 = ' The real storage size is 'Format(STORSIZE,,0)' MB'
record = record||r23||'0d25'x
record = record||'0d25'x
r24 = "----- CPU -----"
record = record||r24||'0d25'x
r25 = ' The CPU model number is 'MODEL
record = record||r25||'0d25'x
r26 = ' The number of online CPUs is 'NUMCPU'.'
record = record||r26||'0d25'x
call cpu_serial
msg.wsock.1=record
return 1
GET_PTR: PROCEDURE
    /* +----------------------------------------+
       | RETURNS A 4 BYTE POINTER AS HEXADECIMAL |
       | STRING AT ADDRESS ADDR+OFFSET.          |
       | ADDR AND OFFSET MUST BE HEX STRINGS.    |
       +----------------------------------------+ */
  ARG ADDR, OFFSET
  TEMP = D2X(X2D(ADDR) + X2D(OFFSET))
  RETURN C2X(STORAGE(TEMP,4))
EXIT
```

```
GET_DATA: PROCEDURE
       /* +----------------------------------------+
          | RETURNS LENGTH BYTES AT ADDR+OFFSET AS  |
          | AN EBCDIC STRING.                       |
          | ADDR AND OFFSET MUST BE HEX STRINGS.    |
          | LENGTH MUST BE A DECIMAL STRING.        |
          +----------------------------------------+ */
    ARG ADDR, OFFSET, LENGTH
    sign = substr(offset,1,1)
    if sign = "-" then
     do
      offset = substr(offset,2)
      TEMP = D2X(X2D(ADDR) - X2D(OFFSET))
      RETURN STORAGE(TEMP,LENGTH)
     end
     else do
           TEMP = D2X(X2D(ADDR) + X2D(OFFSET))
           RETURN STORAGE(TEMP,LENGTH)
          end
EXIT
ipl_timeconv:
ipltime = c2d(ipltime)
IPLTIME = IPLTIME / 100                          /* remove hundreths    */
HH = IPLTIME % 3600                              /* IPL hour            */
MM = (IPLTIME - (3600 * HH)) % 60                /* IPL minute          */
SS = (IPLTIME - (3600 * HH)- (60 * MM)) % 1      /* IPL seconds         */
HH = Right(HH,2,'0')                             /* ensure 2 digit HH   */
MM = Right(MM,2,'0')                             /* ensure 2 digit MM   */
SS = Right(SS,2,'0')                             /* ensure 2 digit SS   */
IPLTIME= HH':'MM':'SS                            /* time format HH:MM   */
return
ipl_dateconv:
IPLDATE= c2d(IPLDATE)
IPLDATE= d2x(IPLDATE)
parse var IPLDATE . 2 IPLDATE 7 .
iplday = date('w',ipldate,'j')
parse value date('s',ipldate,'j') with 1 jjjj 5 mm 7 dd
ipldate= dd"."mm"."jjjj
return
clpa_check:
CLPABYTE  = get_data(ASMVT_PTR,'1',1)         /* point to CLPA byte   */
CHKCLPA  = Bitand(CLPABYTE,'8'x)              /* check for B'1000'    */
CHKCLPA  = C2d(CHKCLPA)                       /* convert to decimal   */
If CHKCLPA < 8 then IPLCLPA = '(with CLPA)'   /* bit off - CLPA       */
  Else IPLCLPA = '(without CLPA)'             /* bit on  - no CLPA    */
return
ipl:
 If Substr(PRODNAME,3,1) < 5 then ,
 IPLADDR  = get_data(RESUCB_PTR,'D',3)        /* point to IPL address */
 Else do
```

```
   IPLADDR  = get_data(CVTSYSAD_PTR,'4',2)   /* point to IPL UCB      */
   IPLADDR  = C2x(IPLADDR)                   /* convert to EBCDIC     */
 End
return
iodf:
 If Substr(PRODNAME,3,1) < 5 then do
   IOCON    = get_data(CVTEXT2_ptr,'6',2)    /* HCD IODFxx or MVSCP   */
                                             /* IOCONFIG ID=xx        */
  r9 = 'The currently active IOCONFIG or HCD IODF is 'IOCON
  r10 = ' '
  r11 = ' '
  r12 = ' '
 End
 Else do
  IODF     = get_data(CDA_ptr,'20',44)
  IODF     = Strip(IODF,T)                    /* del trailing blanks*/
  CONFIGID = get_data(CDA_PTR,'5C',8)         /* point to CONFIG    */
  EDT      = get_data(CDA_PTR,'68',2)         /* point to EDT       */
  IOPROC   = get_data(CDA_PTR,'7C',8)         /* point to IODF Proc */
  IODATE   = get_data(CDA_PTR,'9C',8)         /* point to IODF date */
  IOTIME   = get_data(CDA_PTR,'A4',8)         /* point to IODF time */
  IODESC   = get_data(CDA_PTR,'AC',16)        /* point to IODF desc */
 End
return
master_catalog:
If Substr(FMIDNUM,4,4) <  6604 then do        /* use CAXWA B4 OS390R4 */
  AMCBS_PTR = get_ptr(CVT_PTR,'100')          /* point to AMCBS       */
  ACB_PTR   = get_ptr(AMCBS_PTR,'8')          /* point to ACB         */
  CAXWA_PTR = get_ptr(ACB_PTR,'40')           /* point to CAXWA       */
  MCATDSN  =  get_data(CAXWA_PTR,'34',44)     /* master catalog dsn   */
  MCATDSN  = Strip(MCATDSN,T)                 /* remove trailing blnks*/
  MCATUCB_PTR = get_ptr(CAXWA_PTR,'1C')       /* point to mcat UCB    */
  MCATVOL  = get_data(MCATUCB_PTR,'1C',6)     /* master catalog VOLSER*/
End
Else do                                       /* OS/390 R4 and above  */
  ECVTIPA_PTR = get_ptr(CVTECVT_PTR,'188')    /* point to IPA         */
  IPASCAT = get_data(ECVTIPA_PTR,'E0',63)     /* SYSCAT  card image   */
  MCATDSN  = substr(IPASCAT,11,44)            /* master catalog dsn   */
  MCATDSN  = Strip(MCATDSN,T)                 /* remove trailing blnks*/
  MCATVOL  = Substr(IPASCAT,1,6)
     end
return
os_version:
 If Substr(PRODNAME,3,1) < 6 then do
   r150 = 'The MVS version is 'PRODNAME' - FMID 'FMIDNUM'.'
 end
  Else do
    PRODNAME = get_data(CVTECVT_PTR,'1F0',16) /*point to product name */
    PRODNAME = Strip(PRODNAME,T)              /* del trailing blanks  */
    VER      = get_data(CVTECVT_PTR,'200',2)  /* point to version     */
```

```
    REL      = get_data(CVTECVT_PTR,'202',2)  /* point to release     */
    MOD      = get_data(CVTECVT_PTR,'204',2)  /* point to mod level   */
    VRM      = VER'.'REL'.'MOD
    r150 = ' The OS version is 'PRODNAME VRM' - FMID 'FMIDNUM'.'
  End
return
vtam_version:
VTAMVER  = Substr(ATCVTLVL,3,1)              /* VTAM Version   V     */
VTAMREL  = Substr(ATCVTLVL,4,1)              /* VTAM Release    R    */
VTAMMOD  = Substr(ATCVTLVL,5,1)              /* VTAM Mod Lvl     P   */
If VTAMMOD = ' ' then VTAMLEV = 'V'|| VTAMVER || 'R' || VTAMREL
   else VTAMLEV = 'V' || VTAMVER || 'R' || VTAMREL || 'M' || VTAMMOD
return
cpu_serial:
j = 0
do i=1 to numcpu
  pcca_ptr = get_ptr(pccavt_ptr,j)
  cpuid = get_data(pcca_ptr,'4',12)
  cpuser = substr(cpuid,1,8)
  cpuaddr = get_data(pcca_ptr,'10',2)
  cpuaddr = c2d(cpuaddr)
  r27=' The CPU serial number for CPU 'cpuaddr' is ' cpuid '('cpuser')'
  record = record||r27||'0d25'x
  if i = numcpu  then leave
  j = x2d(j)
  j = j + 4
  j = d2x(j)
end
return
racf:
SECNAM = RCVTID                              /* ACF2 SECNAME = RCVTID*/
If RCVTID = 'RCVT' then SECNAM = 'RACF'      /* RCVT is RACF         */
If RCVTID = 'RTSS' then SECNAM = 'Top Secret'  /* RTSS is Top Secret */
RACFVER  = Substr(RACFVRM,1,1)               /* RACF Version         */
RACFREL  = Substr(RACFVRM,2,2)               /* RACF Release         */
RACFREL  = Format(RACFREL)                   /* Remove leading 0     */
RACFMOD  = Substr(RACFVRM,4,1)               /* RACF MOD level       */
RACFLEV  = RACFVER || '.' || RACFREL || '.' || RACFMOD
If RCVTID = 'RCVT' | RCVTID = 'RTSS' then ,
 RCVTDSN = Strip(get_data(CVTRAC_PTR,'38',44))  /* RACF prim dsn or  */
                                               /* TSS Security File */
If SECNAM = 'RACF' | RACFVRM < '2608' then do
 r190 = ' The security software is 'SECNAM'.' ,
        'The RACF level is 'RACFLEV
  If SECNAM = 'Top Secret' then ,
  r191 = ' The TSS Security File DSN is' RCVTDSN
  If SECNAM = 'RACF' then ,
  r191 = '  The RACF Primary DSN is' RCVTDSN
End
return
```

```
dfp:
If DFAPROD = Ø then do                          /* DFP not DF/SMS       */
  DFAREL   = get_data(CVTDFA_PTR,'2',2)         /* point to DFP release */
  DFPVER   = Substr(DFAREL,1,1)                 /* DFP Version          */
  DFPREL   = Substr(DFAREL,2,1)                 /* DFP Release          */
  DFPMOD   = Substr(DFAREL,3,1)                 /* DFP Mod Lvl          */
  DFPRD    = 'DFP'                              /* product is DFP       */
  DFLEV    = DFPVER || '.' || DFPREL || '.' || DFPMOD
End
Else do                                         /* DFSMS not DFP        */
  DFARELS  = get_ptr(CVTDFA_PTR,'1Ø')           /* point to DF/SMS rel  */
  DFAVER   = X2d(Substr(DFARELS,3,2))           /* DF/SMS Version       */
  DFAREL   = X2d(Substr(DFARELS,5,2))           /* DF/SMS Release       */
  DFAMOD   = X2d(Substr(DFARELS,7,2))           /* DF/SMS Mod Lvl       */
  DFPRD    = 'DFSMS'                            /* product is DF/SMS    */
  DFLEV    = DFAVER || '.' || DFAREL || '.' || DFAMOD
  If DFAPROD = 2 then DFLEV = 'OS/39Ø' DFLEV
  If DFAPROD = 3 then DFLEV = 'z/OS' DFLEV
End
return
tso:
TSVTLVER = get_data(CVTTVT_PTR,'64',1)          /* point to TSO Version */
TSVTLREL = get_data(CVTTVT_PTR,'65',2)          /* point to TSO Release */
TSVTLREL = Format(TSVTLREL)                     /* Remove leading Ø     */
TSVTLMOD = get_data(CVTTVT_PTR,'66',1)          /* point to TSO Mod Lvl */
TSOLEV   = TSVTLVER || '.' || TSVTLREL || '.' || TSVTLMOD
return
```

CLIENT PROGRAM CLIENT

```
/* REXX */
/*****************************************************
This is the basic client REXX.

The program has two targets:

1) Retrieve System info from the servers
2) Send TSO commands to the servers:
   can be used to manage RACF on more systems
   or to send TSO commands to more systems (tso netstat,ping,listc...)

Syntax :
client IP:server_ipaddress PORT:port_number CMD:command

EG:
  client IP:1Ø.1Ø.98.22 PORT:35ØØØ CMD:sysinfo
                                          (To retrieve system info)
  client IP:1Ø.1Ø.98.22 PORT:35ØØØ CMD:TSO lu userid  (RACF command)
  client IP:1Ø.1Ø.98.22 PORT:35ØØØ CMD:TSO netstat
```

```
                                                  (Retrieve TCP/IP info)
  client IP:10.10.98.22 PORT:35000 CMD:shutdown
                                                  (Shutdown the server)
*****************************************************************/
x='SOCKET'('SocketSetStatus')
/***** This call returns the status of the socket set ******
If the socket is connected the result could be :
--> 0 myID Connected Free 17 Used 23
0          -> Return code
myID       -> Socket set ID
Connected -> Socket set is connected
Free 17    -> Number of free sockets in the socket set
Used 23    -> Number of allocated sockets in the socket set
If the socket is NOT connected the result could be :
--> 2005 ESUBTASKNOTACTIVE Subtask not active
This is a error code.
It indicates that the socket it is not allocated
*************************************************************/
IF WORD(x,1)='0' THEN DO
   x='SOCKET'('Terminate')
   END
ARG string
PARSE VAR string ipaddress sport string
call check_input
/* Initialize client control information */
port = sport
/* Initialization */
x='SOCKET'('Initialize','CLICMDS')
/*** This call preallocates the number of sockets in a socket set ***
The result could be : 0 CLICMDS 40 TCP10001
0          -> Return code
CLICMDS    -> Socket set name
40         -> The number of preallocated sockets in a socket set
TCP10001   -> The name of the TCP/IP service
*************************************************************/
IF WORD(x,1)¬='0' THEN DO
   SAY 'Error initializing CLICMDS'
   EXIT
   END
IF ipaddress='NONE' THEN DO
  x='SOCKET'('GetHostId')
/****** This call returns the ipaddress for the current host ****
The result could be : 0 128.228.1.2
0             -> Return code
128.228.1.2   -> Host ipaddress
*************************************************************/
  IF WORD(x,1)¬='0' THEN DO
     SAY 'Error trying to get host id'
     SIGNAL clean_up
     END
```

```
  ELSE ipaddress=WORD(x,2)
  END
/* Initialize for receiving lines sent by the server. */
x = 'SOCKET'('Socket')
/* This call creates an IPv4 socket in the active socket set ****
 * and returns a socket identification ***
The result could be : Ø 1
Ø            -> Return code
1            -> Socket ID
*****************************************************************/
IF WORD(x,1)¬='Ø' THEN DO
   SAY 'Error issuing socket'
   SIGNAL clean_up
   END
/* Pick up the client socket id */
clisock=WORD(x,2)
/* Get the host name */
x='SOCKET'('GetHostName')
/****** This call returns the name of the host ****
The result could be : Ø PROD
Ø            -> Return code
PROD         -> Host name
*****************************************************************/
IF WORD(x,1)¬='Ø' THEN DO
   SAY 'Error getting host name'
   SIGNAL clean_up
   END
hostname = WORD(x,2)
/* Issue af_inet */
x='SOCKET'('Connect',clisock,'AF_INET' port ipaddress)
/****** Tries to establish a connection to another socket ***
The result could be : Ø
Ø            -> Return code
*****************************************************************/
IF WORD(x,1)¬='Ø' THEN DO
   SAY 'Error issuing AF_INET. Rc :' word(x,1)
   if word(x,1) = 61 then say "==> The server is down"
   SIGNAL clean_up
   END
/* Send the information to the server */
x='SOCKET'('Send',clisock,userid() string)
/****** Sends the outgoing data message to a connected socket **
The result could be : Ø 14
Ø            -> Return code
14           -> Length of the data sent
*****************************************************************/
IF WORD(x,1)¬='Ø' THEN DO
   SAY 'Error issuing send'
   SIGNAL clean_up
   END
```

```
/* Wait for lines sent by the server */
DO FOREVER
/* Read the data. Data is returned as a rc len data field */
x='SOCKET'('Read',clisock)
/****** This call reads up to maxlength bytes of data, **
 **     the default is 10,000 **
The result could be : 0 1613 The IPL LOAD PARM used was 3D21SVM1
0            -> Return code
1613         -> Data length
The IPL LOAD PARM used was 3D21SVM1 --> data
*************************************************************/
IF WORD(x,1) ¬='0' THEN DO
   PARSE VAR x . error
   SAY 'Error issuing recv' error
   SIGNAL clean_up
   END
/* allow for the line being null. Abort the connection if it is. */
IF WORD(x,2)='0' THEN LEAVE
/* Get the actual data */
PARSE VAR x . . dataline
DO UNTIL INDEX(dataline,'0D'x)=0
   PARSE VAR dataline trueline '0D'x dataline
   SAY trueline
   END
END
/* Terminate and exit  */
clean_up:
x='SOCKET'('Terminate','CLICMDS')
/***** This call closes all sockets in the socket set ******
The result could be : 0 CLICMDS
0         -> Return code
CLICMDS   -> Subtask Id
*************************************************************/
RETURN
check_input:
ippre=substr(ipaddress,1,3)
portpre=substr(sport,1,5)
cmdpre=substr(string,1,4)
ipaddress = strip(substr(ipaddress,4))
sport = strip(substr(sport,6))
string = strip(substr(string,5))
if ipaddress = ' ' | sport = ' ' | string = ' ' ,
  | ippre ¬= 'IP:' | portpre ¬= 'PORT:' ,
   | cmdpre ¬= 'CMD:'
   then do
          say " "
          say " "
           say "*************************************************"
          say "  Invalid Syntax.The correct syntax is:"
          say "  IP:ipaddress PORT:port_number CMD:string"
```

```
          say "  Example -> IP:10.10.10.3 PORT:1946 CMD:sysinfo"
          say "************************************************"
          say " "
         exit
         end
return
```

INSTALLATION

The programs are compatible with z/OS 1.4 and above. To install the programs do the following:

- Copy the program srvcmds to the user library alias.user.lib.
- Copy the program client to the same user library, alias.user.lib.
- Copy the JCL streams, jclicmds and jsrvcmds, to the same user library, alias.user.lib.

Alias.user.lib is the user library.

The JCL for server start-up is:

```
//useridSR JOB (TSO,SIG),'.JSERVER.',MSGCLASS=R,
//           NOTIFY=userid,REGION=8M,CLASS=A,TIME=1440
//*
//A EXEC   PGM=IKJEFT01,DYNAMNBR=50,REGION=6M
//SYSPROC  DD DSN=alias.user.lib,DISP=SHR
//SYSTCPD  DD DSN=alias.user.lib(TCPIPPAR),DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSTSOUT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN  DD *
PROFILE NOPREFIX
SRVCMDS 1957
```

SRVCMDS is the name of the REXX server program in the library alias.user.lib.

TCPIPPAR is a copy of the configuration that is allocated to the DD SYSTCPD in the TCP/IP start-up procedure.

TCP/IP start-up procedure:

```
//TCP0001 PROC PARMS='CTRACE(CTIEZB00)'
//*
```

```
//TCPIP    EXEC PGM=EZBTCPIP,
//             PARM='&PARMS',
//             REGION=7500K,TIME=1440
//*
//STEPLIB  DD DISP=SHR,DSN=TCP001.LOAD
//*
//SYSPRINT DD SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//ALGPRINT DD SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//SYSOUT   DD SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//CEEDUMP  DD SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//SYSERROR DD SYSOUT=*
//*
//PROFILE  DD DISP=SHR,DSN=TCP001.LIB.PARM(PROFILE)
//*
//SYSTCPD  DD DSN=TCP001.LIB.PARM(PRMPROD),DISP=SHR
```

In this example, the member PRMPROD (DD SYSTCPD) will be copied to alias.user.lib(TCPIPPAR).

The command syntax is:

```
srvcmds port_number
```

Here is an example of the server log:

```
*************************************************************
PROFILE NOPREFIX
READY
SRVCMDS 1957
SRVCMDS: initializing ....
SRVCMDS: initialized: ipaddress=10.10.10.31 port=1957
User userid issued command TSO LISTC ENT('userid.LIB.JCL') ALL at
15:07:03 30/03/05
User userid issued command SYSINFO at 15:08:18 30/03/05
User userid issued command TSO LU userid at 15:08:18 30/03/05
User userid issued command SYSINFO at 16:39:20 30/03/05
User userid issued command SYSINFO at 08:43:13 31/03/05
User userid issued command SYSINFO at 08:44:13 31/03/05
User userid issued command SYSINFO at 08:51:38 31/03/05
User userid issued command SYSINFO at 08:52:07 31/03/05
User userid issued command SYSINFO at 09:42:39 31/03/05
User userid issued command TSO NETSTAT at 11:17:15 31/03/05
*************************************************************
```

The JCL for client start up is:

```
//aliasCL JOB (TSO,SIG),'.JSERVER.',MSGCLASS=R,
//          NOTIFY=alias,REGION=8M,CLASS=A,TIME=1440
//*
//*********** Samples *****************
```

```
//*
//*** SERVER shutdown  ***
//* CLIENT IP:10.10.10.31 PORT:3000 CMD:SHUTDOWN
//*
//*** TSO Userid files list  ***
//* CLIENT IP:10.10.10.31 PORT:2500 CMD:TSO LISTC LEV(userid)
//*
//*** RACF Commands ***
//* CLIENT IP:10.10.10.31 PORT:3400 CMD:TSO LU userid
//* CLIENT IP:10.10.10.31 PORT:2500 CMD:TSO ALU userid
PASSWORD(password) RESUME
//*
//*** OMVS environment commands ***
//* CLIENT IP:172.22.64.31 PORT:6000 CMD:TSO NETSTAT
//*
//A EXEC PGM=IKJEFT01,DYNAMNBR=50,REGION=6M
//SYSPROC DD DSN=alias.user.lib,DISP=SHR
//SYSTCPD  DD DSN=alias.user.lib(TCPIPPAR),DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSTSOUT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSIN DD *
PROFILE NOPREFIX
CLIENT IP:10.10.10.31 PORT:1957 CMD:SYSINFO
CLIENT IP:10.10.10.31 PORT:1957 CMD:TSO LU userid
```

CLIENT is the name of the REXX client program in the library alias.user.lib.

The command syntax is:

```
client IP:server_ipaddress PORT:port_number CMD:command
```

An example of using the SYSINFO command:

```
   This command obtains the system report from the server.

------------------------------------------------------------------
System : PROD          Lparname : A2ZOS
------------------------------------------------------------------


Today is Thursday 31 Mar 2005 (2005.090)

----- IPL -----
  The last Ipl was 07.03.2005 at 21:18:06  (with CLPA)
  The system IPL address was 8262 volume: R4TA1B
  The IPL LOAD PARM used was 3D21SVM1 (3D21 SV M 1)
```

```
----- Sysplex -----
  The Sysplex name is : SVILPLEX

----- IODF -----

   The currently active IODF data set is SYS000.IODF05
    Configuration ID = SVILOS    EDT ID = 25
    TOKEN: Processor  Date      Time      Description
           SAR0A      05-03-18  17:03:41  SYS000  IODF03

----- Master catalog -----
  The Master Catalog is CATALOG.ICFM.VCATSA1 on CATSA1.

----- System Software -----
  The OS version is z/OS 01.04.00 - FMID HBB7707.

  The VTAM Level is V6R1M4.

  The primary job entry subsystem is JES2

 The GRS system id is PROD

 The SMF system id is PROD

 The security software is RACF. The RACF level is 7.70.7
  The RACF Primary DSN is SYS1.RACFPRM.SV00

 The DFSMS level is z/OS 1.3.0.

 The TSO level is 3.3.3

----- Real Storage Size -----
 The real storage size is 13312 MB

----- CPU -----
 The CPU model number is 2084
 The number of online CPUs is 0004.
 The CPU serial number for CPU 0 is  0002094E2084 (0002094E)
 The CPU serial number for CPU 1 is  0002094E2084 (0002094E)
 The CPU serial number for CPU 2 is  0002094E2084 (0002094E)
 The CPU serial number for CPU 3 is  0002094E2084    (0002094E)
```

The SHUTDOWN command can be used to execute server shutdown.

Examples of TSO commands that can be used are:

- TSO:
  - ALLOC

  - LIST
  - LISTALC
  - LISTBC
  - LISTCAT
  - LISTDS
  - RENAME
  - PARMLIB
  - SYNC
  - SEND
  - RECEIVE
  - TRANSMIT.
- OMVS:
  - NETSTAT
  - PING
  - NSLOOKUP.
- RACF:
  - LU
  - ALU.

And so on.

This command can be used in many different situations. For example:

- System information:

```
CLIENT IP:10.10.10.31 PORT:1957  CMD:SYSINFO
```

- RACF command:

```
CLIENT IP:10.10.10.35 PORT:2345  CMD:TSO LU userid
```

- OMVS command:

```
CLIENT IP:10.10.10.90 PORT:35000 CMD:TSO netstat      ()
```

The examples show how it is possible to execute different commands on many servers at the same time.

Magni Mauro
Systems Engineer (Italy)

Useful MVS and VTAM commands

Sometimes people get so used to using their TCP/IP monitoring software that they forget that it's very easy to get information about the network and to control what's going on by using old-fashioned console commands. I thought it might be useful simply to remind people of some of them:

- To list the names and status of TCP/IP stacks:

```
D TCPIP
```

- To display a list of TCP/IP display options:

```
D TCPIP,{procname},HELP
```

- To display a list of TCP/IP vary options:

```
V TCPIP,{procname},HELP
```

- To display socket information for the TCP/IP stack:

```
D TCPIP,{procname},Netstat,ALLCONN|CONN
```

- To display device and link status for the TCP/IP stack:

```
D TCPIP,{procname},Netstat,DEVlinks
```

- To display the IP address(es) for the TCP/IP stack:

```
D TCPIP,{procname},Netstat,HOME
```

- To display the routing table for the TCP/IP stack:

```
D TCPIP,{procname},Netstat,ROUTE
```

- To start or stop the device name identified in the NETSTAT DEV output:

```
V TCPIP,{procname},START|STOP,devname
```

- To display the contents of the ARP cache for the TCP/IP stack:

```
D TCPIP,{procname},Netstat,ARP
```

- To purge the ARP cache for the specified adapter (*linkname* is found from the NETSTAT,DEVLINKS command):

```
V TCPIP,{procname},PURGECACHE,linkname
```

- To perform a specified function for TELNET:

```
V TCPIP,{procname},Telnet,xxxx
```

Useful VTAM commands that relate to OSA cards include:

- To display the network named in the ID field:

```
D NET,ID=name
```

  Extra parameters may be added to give extended information about the node. These include:

```
,SCOPE=ONLY|ACT|ALL|INACT,E
```

- To show the status of all the active major nodes or applications:

```
D NET,MAJNODES|APPLS
```

- To list all the nodes in pending state:

```
D NET,PENDING
```

- To activate the VTAM resource identified by the *name*:

```
V NET,ACT,ID=name
```

- To inactivate the VTAM resource identified by the *name*:

```
V NET,INACT,ID=name
```

I hope this quick reminder helps with solving any problems you meet in the future.

TCP/IP connections reporting

INTRODUCTION

TCP/IP on the mainframe has been growing in importance over the last few years since the advent of the Internet and enterprise-based networks. The zSeries platform provides an environment on which critical business applications flourish. The demands placed on these systems are ever-increasing and such demands require a solid, scalable, highly-available, and high-performing operating system and TCP/IP component. z/OS and Communications Server for z/OS provide for such a requirement with a TCP/IP stack that is robust and rich in functionality. Since every TCP/IP environment is different, optimum performance can be achieved only when the system is tuned to match its specific environment. The performance of Communications Server for z/OS can be improved significantly by proper tuning. TCP/IP performance is influenced by a number of parameters that can be tailored for the specific operating environment. These include not only TCP/IP stack performance, which benefits all applications using the stack, but also applications shipped with Communications Server for z/OS IP, such as TN3270 and FTP.

The need to manage mainframe networks has been with us for a long time. Consequently, many of the tools we use have a long history and in some cases they may have become less effective in managing today's modern networks. On the other hand, mainframe TCP/IP networking is invariably complex, often challenging, and sometimes a bit confounding. This is to

be expected given what can be involved and what is at stake. Mainframe TCP/IP networking, as we know, can involve multiple stacks per LPAR, virtual addresses, gigabit OSA interfaces, hypersockets, Dynamic VIPA takeovers across a sysplex, disparate application protocols, many hosts, and a lot of connectionless interactions. There is also a need to be well aware of routers, switches, and firewalls – with performance, in particular '3270' response times, always a concern, plus the ever-present security worry. To stay on top of all of this, to deliver 'zero downtime' operations, you need a good mainframe network monitor that is probing, incisive, and thorough and that, moreover, works in true real-time. In a TCP/IP environment, real-time monitoring is implemented using the SNMP protocol, and is based on internal variables that are maintained by SNMP subagents. But on z/OS, a lot of the information that is written in SMF records is useful from a real-time monitoring perspective too.

COLLECTING PERFORMANCE-RELATED DATA

In general, an installation may use Systems Management Facilities (SMF) records for various purposes such as performance management, auditing, and capacity planning, as well as for accounting. In general, SMF records are created for deferred processing and analysis. The z/OS V1R2 Communications Server provides enhanced TCP/IP SMF recording, including a new standard TCP/IP SMF record format and several new TCP/IP SMF records. What this enhancement means is that all TCP/IP SMF records now use a new standard record format. TCP/IP now provides a significant amount of new data in several new SMF records that can be used for an installation's capacity planning, tuning, and/or accounting procedures. These new records include:

- Server statistics for TCP and UDP applications on an SMF interval basis. This new record can be used to determine workload levels of key TCP/IP servers during different intervals. This information can be helpful in capacity planning and tuning.

- UDP statistics are now available in a new SMF record. This record can be used to determine the level of UDP activity for key UDP client or server applications.
- Network device layer statistics are now available in a new SMF record that is created on an SMF interval basis. This record can be very useful in capacity planning and tuning of IP workloads.

There can be multiple uses for given SMF records. SMF records can be cut at multiple levels in the TCP/IP protocol stack, and the type of information that can be included depends on where the SMF record is created:

- At the IP and interface layer – we know about ICMP activity, IP packet fragmentation and reassembly activity, and IP checksum errors. From a performance and capacity planning point of view, this information is of interest because it allows the installation to aggregate network-layer activity to physical interfaces, which is an important aspect of both performance and capacity management.
- At the transport protocol layer – we know about IP addresses, port numbers, and host names. For TCP-related work, we know about connections and information that is related to TCP connections, such as byte counts, connection times, reliability metrics, and performance metrics. For UDP-related work, each UDP datagram is a separate entity, and the only way we can aggregate information for UDP is on a UDP socket-level, where we could cut SMF records every time a UDP socket is closed.
- At the application layer – we know more details about what goes on, but every application is different and they require separate SMF record definitions and the ability to write those SMF records in order to implement application-layer SMF recording. We currently do it for the stack telnet server and the FTP server, but not for any other servers.

To have all possible TCP/IP-related SMF records created, you may use the SMFCONFIG statement to log the use of TCP by

applications using SMF log records. Using SMFCONFIG to turn on SMF logging allows you to request that standard subtypes be assigned to the TCP/IP SMF records. The SMFPARMS statement provides a similar capability but requires the installation to select the subtype numbers to be used. Use of the SMFCONFIG parameter is recommended instead of SMFPARMS. An important warning is needed here: if you choose to collect all possible record subtypes, you will incur unnecessary system overhead and system performance degradation. Therefore I would recommend that you plan carefully what you need to collect data about and exercise strict control over the collection of these records through the use of the SMFCONFIG and TELNETPARMS statements in PROFILE.TCPIP or through various statements in the FTP.DATA file. The good news is that you can turn SMF recording on and off dynamically by the use of the VARY TCPIP,,OBEYFILE command – if you want to change the TCP/IP configuration without stopping and starting the TCP/IP address space, you can dynamically change many of the TCP/IP configuration options established by the PROFILE.TCPIP dataset. To do this, put the changed configuration statements in a separate dataset and process it with the VARY TCPIP,,OBEYFILE command.

GATHERING CONNECTIONS STATISTICS

By using the TCPTERM start option in the SMFCONFIG start-up parameters you can request that SMF records of subtype 2 are created when TCP connections are terminated. The TCP connection termination record is collected whenever a TCP connection is closed or aborted. This record contains all pertinent information about the connection, such as elapsed time, bytes transferred, and so on. Please note that because this information duplicates all the information contained in the TCP connection initiation record, it is recommended that users collect only the TCP connection termination record. In addition to that, since this record is generated for every single TCP connection, this can generate significant load on a server

and rapidly fill the SMF datasets. Care should be exercised in its use.

The collected data that will help you in TCP/IP performance management

As we know, performance management includes the tasks related to verifying that defined service levels are met, and, if they are not, identifying possible causes. Some examples of potential reports related to performance management are:

- TCP connection elapsed time per server port number per time of day (potentially broken down by source IP address).
- Number of TCP connections per server port number per time of day (potentially broken down by source IP address).
- Number of inbound/outbound bytes transferred in TCP connections per time of day (potentially broken down in various ways: by destination or source port, by destination IP address, or in total, etc).
- TCP retransmission activity per time of day (potentially broken down by destination IP address).
- Number of outbound TCP connections per time of day (potentially broken down by destination IP address).

From a capacity planning point of view, you can obtain the total number of TCP connections per reserved server port number per day and generate an analysis of the average and variations around the average during daily peak periods.

In addition to that, SMF records of subtype 2 can be used for auditing, which involves tasks that are related to identifying and proving that individual events have taken place. Some examples of potential reports related to auditing are:

- Detailed information about specific TCP connections, IP addresses, server/client identification, duration, number of bytes, etc.

- Details about activity that involves a specific client or server.

Finally, these records can be used for accounting, which involves tasks that are related to calculating how much each individual user or organizational unit should be charged for use of the shared central IS resources. Input to such calculations varies, but is often based on data quantities or bandwidth usage. For TCP/IP, additional metrics may be defined, such as type of service used (FTP, LPD, Web server, etc), and TCP connection-related information (number of connections, duration, byte transfer counts, etc). Some examples of potential reports related to accounting are:

- Aggregated number of connections to a given server from a given source in terms of a specific client IP address.
- Accumulated connect time to a given server from a given source in terms of a specific client IP address.
- Number of bytes transferred to or from a given source in terms of a specific client IP address.
- Application-level accounting information specific to each individual server, for example:
  - for FTP, number of transfer operations and bytes retrieved or stored per user ID.
  - for TN3270, number of sessions and session-type (TN3270/TN3270E/LINEMODE).

In order to provide a starting point from which one can begin to measure and analyse TCP/IP connections I have coded a sample report writer. Two reports are generated by this report writer.

The first report is a TCP connection termination report, which is made up of five parts. The first part is the system identification part. The second part (connection termination detail report) provides users with an extensive overview of all recorded connections in detail, and can be used to view a performance

summary of each TCP/IP connection. There are several indicators of connection/network performance, among which are round-trip time, retransmissions counter, MaxRate, and congestion window size.

Using the round-trip time and round-trip variance attributes, for example, can help you determine the cause of network congestion, whether it be host or route-related. Round-trip variance is the statistical variation of round-trip times since the connection was established. Since round-trip time is the trip time of the last segment sent, it is only meaningful in relation to round-trip variance. Used together, however, these values can indicate how healthy the route is through the network from sending host to receiving host and back. A small variance indicates that the current round-trip time can be used as an average. A high variance may indicate either that the current round-trip time may not represent the average or that round-trip times are sporadic. You may not want to rely on variance for short sessions, or for a session that just started, since in either case few segments were transmitted. Variance is more accurate over longer, more robust sessions. If the problem is not route-related but most of the IP traffic through this address space is experiencing it, you may have to look into possible z/OS system performance problems.

MaxRate indicator is obtained by taking the congestion window and dividing it by the round-trip time, and it is an integer value representing the maximum rate in kilobits per second (Kbps) allowed for traffic in this service class. This attribute is valid only for TCP. If not specified, or specified as 0, there will not be any enforcement of the maximum rate of a connection by the local host. If a number other than 0 is specified, each TCP connection that is mapped to this PolicyAction will have its rate limited to this MaxRate. Enforcement of the MaxRate is performed by the TCP/IP stack by adjusting the TCP congestion window based on the connection round-trip time. Because the minimum of the congestion window is one TCP segment size, the minimum MaxRate that can be enforced is one TCP

segment over the round-trip time. If a TCP connection has a very small round trip delay and traverses over a very high bandwidth network (for example Gbit Ethernet LAN), the minimum rate that this TCP connection can send (one segment per round-trip time) can be high. Therefore, users and network-administrators need to know their network characteristics when setting this MaxRate; it may not be enforceable if the minimum TCP rate (for example one segment over round-trip time) already exceeds this specified MaxRate. As noted, TCP segment size can play a role in this TCP minimum rate; that is, for a given round-trip delay, the larger the segment size, the higher the minimum TCP rate. Different factors can affect the TCP segment size, for example the local MTU size definition, the Path MTU discovery flow (this mechanism is used to discover the maximum MTU size that can be sent into the network without resulting in IP fragmentation), the receiver's maximum segment size, and so on.

The third part of this report is the first exception report and it deals with connections with retransmissions (which lower TCP/IP performance) above a certain limit. In this case, I have set our limit to 5. You can specify a different retransmissions limit. The fourth part of this report is the second exception report and it lists all connections with a round-trip time of >400ms (you can specify a different limit). The last part of this report lists connections with very large congestion window sizes.

The second report, TCP connection termination statistics report, is a summary report and it provides users with the aggregated number of connections, connect time, TCP retransmission activity, inbound/outbound bytes transferred in TCP connections, and total number of TCP connections per reserved server port number.

CODE

Based on TCP/IP SMF record descriptions obtained from the *z/OS Communications Server: IP Configuration Reference*

Version 1 Release 2 (SC31-8776-02) manual, a sample TCP/IP connections report writer was written. In order to extract TCP/IP connection termination event information from SMF data, I have constructed a three-part job stream. In the first step (CONDMP) SMF records 119 subtype 2 are extracted from the SMF dataset to a file that can be used as a base for archived records. In the second step (CONSEL), previously extracted records (selection being defined by INCLUDE's condition) are sorted and copied to a file that is the input to analysis and reporting TCPCONN REXX EXEC invoked in the last (CONREXX) step.

Sample JCL to execute SMF type 119 data extract and TCP/IP connection reporting:

```
//*-------------------------------------------------------------------*
//* UNLOAD SMF 119 subtype 2 RECORDS FROM VSAM OR VBS                 *
//* Note: change the DUMPIN DSN=your.smfdata to the name of           *
//* the dataset where you currently have SMF data being               *
//* recorded. It may be either an SMF weekly dataset or an active     *
//* dump dataset. If you chose the latter, then prior to              *
//* executing this job, you need to terminate SMF recording           *
//* to the currently active dump dataset to allow the                 *
//* unload of SMF records.                                            *
//* Also, change the DCB reference to match the name of your          *
//* weekly SMF dump dataset. Change the SPACE allocation values       *
//*-------------------------------------------------------------------*
//CONDMP   EXEC PGM=IFASMFDP,REGION=0M
//INDA1 DD DSN=your.smfdata,DISP=SHR
//OUTDA DD DSN=&&COPY,DISP=(NEW,PASS),
//         UNIT=SYSDA,SPACE=(CYL,(a,b),RLSE),
//         DCB=(your.weekly.smfdata)
//SYSPRINT DD SYSOUT=X
//SYSIN    DD *
       DATE(2005135,2005178)
       INDD(INDA1,OPTIONS(DUMP))
       OUTDD(OUTDA,TYPE(119(2)))
/*
//*-------------------------------------------------------------------*
//* COPY VBS TO VB, DROP HEADER/TRAILER RECORDS, SORT ON DATE/TIME    *
//* Note: change the CONSMF DSN=hlq.conrec to the name of             *
//* the dataset you'll use in the last step.                          *
//* Change the SPACE allocation values.                               *
//*-------------------------------------------------------------------*
//CONSEL  EXEC PGM=ICETOOL
//DFSMSG  DD SYSOUT=*
//RAWSMF  DD DSN=&&COPY,DISP=(OLD,DELETE)
```

```
//CONSMF  DD DSN=hlq.conrec,DISP=(NEW,CATLG),
//           DCB=(RECFM=VB,LRECL=32756,BLKSIZE=Ø),UNIT=SYSDA,
//           SPACE=(CYL,(x,y),RLSE)
//CON1CNTL DD *
  OPTION DYNALLOC,VLSHRT,SPANINC=RC4
  INCLUDE COND=(6,1,BI,EQ,119,AND,23,2,BI,EQ,2)
  SORT FIELDS=(11,4,PD,A,7,4,BI,A)
/*
//TOOLMSG DD SYSOUT=*
//REPORT  DD SYSOUT=*
//TOOLIN  DD *
  SORT FROM(RAWSMF) TO(CONSMF) USING(CON1)
/*
//*-------------------------------------------------------------*
//* FORMAT TCP/IP Connection termination TYPE 119 subtype 2 records*
//* Note: change the SYSEXEC DSN=your.rexx.library to the name     *
//* of the dataset where you have placed the TCPCONN REXX EXEC.    *
//* Also, change the SMFREC DSN=hlq.conrec to the name of          *
//* the dataset you have created in the previous step.             *
//*-------------------------------------------------------------*
//CONREXX  EXEC PGM=IKJEFTØ1,REGION=ØM,DYNAMNBR=5Ø
//SYSEXEC  DD DSN=SYSTMØ5.RC.CLIST,DISP=SHR
//SMFREC   DD DSN=SYSTMØ5.SMF119.RECØØX2,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
prof nopref
%TCPCONN
/*
```

TCPCONN reporting EXEC:

```
/* REXX TCP/IP subtype Ø2   ------------------------------------ */
/* The TCP Connection Termination record is collected whenever     */
/* a TCP connection is closed or aborted. This record contains     */
/* all pertinent information about the connection, such as elapsed */
/* time, bytes transferred, and so on.                             */
/*----------  ---------------------------------------------------*/
Numeric digits 32
ADDRESS TSO
userid=SYSVAR(SYSUID)
/*-------------------------------------------------------------*/
/* Part 1: Handle report files allocation & datasets         */
/*         existence                                         */
/*-------------------------------------------------------------*/
conc = userid||'.conc.rep'   /* Connection close report      */
cons = userid||'.cons.rep'   /* Connection close stat report */
x = MSG('ON')
IF SYSDSN(conc) = 'OK'
   THEN "DELETE "conc" PURGE"
"ALLOC FILE(CONN) DA("conc")",
```

```
   " UNIT(SYSALLDA) NEW TRACKS SPACE(29,29) CATALOG",
   " REUSE RELEASE LRECL(275) RECFM(F B)"
IF SYSDSN(cons) = 'OK'
   THEN "DELETE "cons" PURGE"
"ALLOC FILE(CONNS) DA("cons")",
   " UNIT(SYSALLDA) NEW TRACKS SPACE(29,29) CATALOG",
   " REUSE RELEASE LRECL(115) RECFM(F B)"
/*-----------------------------------------------------------*/
/*  Print TCP Connection Termination report header          */
/*-----------------------------------------------------------*/
hdr.1 = left('TCP Connection Termination report',5Ø)
hdr.2 = left(' ',1,' ')
hdr.3  =left('Report produced on',18),
        ||left(' ',1,' ')||left(date(),12),
        ||left('at',3,' ')||left(time(),1Ø)
hdr.4 = left(' ',1,' ')
  "EXECIO * DISKW CONN(STEM hdr.)"
hd.1 = left("PART 2: Connection termination detail report:",5Ø)
hd.2 = left(' ',1,' ')
hd.3 = left("Connection ended at:",24)
hd.4 = left("Date",12)       left("time",14)        ,
       left("Duration",12)   left("Dur.sec",7)      ,
       left("ASName",8)      left("Sec. ID",7)      ,
       left("ASID",5)        left("User ID",8)      ,
       left("Status",7)      left("Conn ID",7)      ,
       left("Protocol",8)    left("Application",23) ,
       left("LIP",9)         left("LPN",7)          ,
       left("RIP",9)         left("RPN",4)          ,
       left("RTT",4)         left("RVA",6)          ,
       left("M.R",7)         left("InB",3)          ,
       left("InB.sec",7 )    left("OutB",4)         ,
       left("OutB.sec",8 )   left("InSG",6)         ,
       left("OutSG",5)       left("SGsize",6)       ,
       left("Rxmit",5)       left("WSend",7)        ,
       left("WMax",8)        left("WCong",6)        ,
       left("SWidx",7)       left("CWidx",6)
hd.5 = left('-',272,'-')
/*-----------------------------------------------------------*/
/*  Print TCP Connection Termination statistics report hd. */
/*-----------------------------------------------------------*/
sdr.1 = left('TCP Connection Termination statistics report',8Ø)
sdr.2 = left(' ',1,' ')
sdr.3  =left('Report produced on',18),
        ||left(' ',1,' ')||left(date(),12),
        ||left('at',3,' ')||left(time(),1Ø)
sdr.4 = left(' ',1,' ')
  "EXECIO * DISKW CONNS(STEM sdr.)"
hs.1 = left(' ',1,' ')
hs.2 = left("Port",4)         left("Protocol",8),
       left("Application",18) left("used",4)    ,
```

```
       left("Conn.time",11)   left("Rxmit",9)   ,
       left("InB",8)          left("OutB",6)    ,
       left("InSG",5)         left("OutSG",11)  ,
       left("InB %",11)       left("OutB %",13)
hs.3 = left('-',112,'-')
  "EXECIO * DISKW CONNS(STEM hs.)"
xhd.1 = hd.2
xhd.2 = left("PART 3: Connections with retransmissions > 5:",60)
xhd.3 = hd.2; xhd.4 = hd.3
xhd.5 = hd.4; xhd.6 = hd.5
rhd.1 = hd.2
rhd.2 = left("PART 4: Connections with Round Trip Time > 400 ms.",60)
rhd.3 = hd.2; rhd.4 = hd.3
rhd.5 = hd.4; rhd.6 = hd.5
chd.1 = hd.2
chd.2 = left("PART 5: Connections with congestion window > 10 max.",70)
chd.3 = hd.2; chd.4 = hd.3
chd.5 = hd.4; chd.6 = hd.5
/* ---------------------------------------------------------*/
/* Set various counters to zero                              */
/* ---------------------------------------------------------*/
inrec = 0; xmit  = 0; totinsg   = 0
rtt   = 0; cw    = 0; totoutsg  = 0
port_count.= 0; port_time.= 0
port_inb.  = 0; totinb    = 0
port_outb. = 0; totoutb   = 0
port_ins.  = 0; port_xmm. = 0
port_outs. = 0; Tot_elap  = 0
Tot_xmit   = 0; Tot_con   = 0
 /*---------------------------------------------------------*/
 /*                 Main processing loop                    */
 /*---------------------------------------------------------*/
 DO FOREVER
   "EXECIO 1 DISKR SMFREC"
    IF RC ¬= 0 THEN call End_of_file
 else do
    inrec = inrec + 1
    PARSE PULL record
    PARSE VAR record header 41 rest
SMF119HDType= c2d(substr(header,2,1))       /* Record type */
 call SMFREC_header
  end
 End
/*---------------------------------------------------------*/
/* Print "End of file" summary stat                        */
/*---------------------------------------------------------*/
End_of_file:
Select
 when xmit > 0 then do
 "EXECIO * DISKW CONN (STEM xhd.)"
```

```
 "EXECIO * DISKW CONN (STEM  xm.)"
  end
 otherwise nop
End
Select
 when rtt > Ø then do
  "EXECIO * DISKW CONN (STEM  rhd.)"
  "EXECIO * DISKW CONN (STEM trip.)"
 end
 otherwise nop
End
Select
 when cw > Ø then do
  "EXECIO * DISKW CONN (STEM  chd.)"
  "EXECIO * DISKW CONN (STEM conw.)"
 end
 otherwise nop
End
lgg.1 = "                                                                "
lgg.2 = "LEGEND:                                                         "
lgg.3 = "                                                                "
lgg.4 = "End date   - Connection end date: connection entered            "
lgg.5 = "             TIMEWAIT or LASTACK state.                         "
lgg.6 = "end time   - Time of connection end.                            "
lgg.7 = "Duration   - Connection duration in hh:mm:ss.tt format.         "
lgg.8 = "Dur.sec    - Connection duration in seconds.                    "
lgg.9 = "ASName     - Started task qualifier or address space name of    "
lgg.1Ø= "             address space that writes this SMF record.         "
lgg.11= "Sec. ID    - User ID of security context under which this       "
lgg.12= "             SMF record is written.                             "
lgg.13= "ASID       - ASID of address space that writes this SMF rec.    "
lgg.14= "User ID    - TCP socket resource name: address space name of    "
lgg.15= "             address space that closed this TCP connection.     "
lgg.16= "Status     - Socket status:                                     "
lgg.17= "             Passive Open (server socket)                       "
lgg.18= "             Active Open  (client socket)                       "
lgg.19= "Conn ID    - TCP socket resource ID (Connection ID).            "
lgg.2Ø= "LIP        - Local IP address at time of connection close.      "
lgg.21= "LPN        - Local port number at time of connection close.     "
lgg.22= "RIP        - Remote IP address at time of connection close.     "
lgg.23= "RPN        - Remote port number at time of connection close.    "
lgg.24= "RTT        - Round Trip Time in milliseconds.                   "
lgg.25= "RVA        - Round Trip Time Variance.                          "
lgg.26= "M.R        - Max Rate of traffic in this service class.         "
lgg.27= "InB        - Inbound byte count.                                "
lgg.28= "InB.sec    - Inbound byte per second.                           "
lgg.29= "OutB       - Outbound byte count.                               "
lgg.3Ø= "OutB.sec   - Outbound byte per second.                          "
lgg.31= "InSG       - Inbound segment count.                             "
lgg.32= "OutSG      - Outbound segment count.                            "
```

```
lgg.33= "SGsize     - Send Segment Size.                                  "
lgg.34= "Rxmit      - Number of times retransmission was required.        "
lgg.35= "WSend      - Send Window Size at time of connection close.       "
lgg.36= "WMax       - Maximum Send Window Size.                           "
lgg.37= "WCong      - Congestion Window Size at time of close.            "
lgg.38= "SWidx      - Send Window index (ratio to Maximum Send Window)    "
lgg.39= "CWidx      - Congestion Window index (vs Maximum Send Window)    "
  "EXECIO * DISKW CONN (STEM lgg.)"
slg.1 = "                                                                 "
slg.2 = "LEGEND:                                                          "
slg.3 = "                                                                 "
slg.4 = "Application - The name of application using the port             "
slg.5 = "Used        - The number of connection using this port           "
slg.6 = "Conn.time   - Total connection time for this port                "
slg.7 = "Rxmit       - Total number of retransmission for this port       "
slg.8 = "InB         - Total Inbound byte count for this port             "
slg.9 = "OutB        - Total Outbound byte count fort this port           "
slg.10= "InSG        - Total Inbound segment count for this poer          "
slg.11= "OutSG       - Total Outbound segment count fort this port        "
slg.12= "InB %       - Pct of total Inbound byte count for this port      "
slg.13= "OutB %      - Pct of total Outbound byte count fort this port    "
 ss.1 = left('-',88,'-')
 DO lpn = 1 to 9999
 if port_count.lpn > 0 then
 call printline right(lpn,4)      ,     /* Port number */
          left(Pxlate(lpn),26)    ,     /* Port description            */
          right(port_count.lpn,4),      /* Number of port connections  */
          smf(port_time.lpn)      ,     /* Connection time             */
          right(port_xmm.lpn,4)   ,     /* Retransmissions             */
          right(port_inb.lpn,9)   ,     /* Inbound byte count          */
          right(port_outb.lpn,9)  ,     /* Total Outbound byte count   */
          right(port_ins.lpn,6)   ,     /* Inbound segment count       */
          right(port_outs.lpn,6)  ,     /* Outbound segment count      */
          format((port_inb.lpn/totinb)*100,6,4) ,
          format((port_outb.lpn/totoutb)*100,6,4)
  end
  "EXECIO * DISKW CONNS(STEM ss.)"
 call printline left("TOTAL:",31)  right(Tot_con,4)    ,
                smf(Tot_elap)      right(Tot_xmit,4)   ,
                right(totinb,9)    right(totoutb,9)    ,
                right(totinsg,6)   right(totoutsg,6)
  "EXECIO * DISKW CONNS(STEM slg.)"
 /* Close & free all allocated files  */
 "EXECIO 0 DISKR CONN   (FINIS"
 "EXECIO 0 DISKR CONNS  (FINIS"
 "EXECIO 0 DISKR SMFREC (FINIS"
 say "TCP Connection report file..................."conc
 say "TCP Connection Statistic report file........."cons
 "FREE FILE(CONN SMFREC)"
 EXIT 0
```

```
SMFREC_header:
/*-------------------------------------------------------------------*/
/* Part 2: Get SMF record header segment                             */
/*-------------------------------------------------------------------*/
s119HDType= c2d(substr(header,2,1))             /* Record type       */
s119HDTime = SMF(c2d(substr(header,Ø3,Ø4)))     /* Record written time*/
s119HDDate ,
       = substr(c2x(substr(header,Ø7,Ø4)),3,5)  /* Record written date*/
s119HDSID = substr(header,11,Ø4)                /* System id         */
s119HDSSI = substr(header,15,Ø2)                /* Subsystem ID      */
SubType = c2d(substr(header,19,Ø2))
s119SD_TRN = c2d(substr(header,21,Ø2))
s119IDOff  = c2d(substr(header,25,Ø4))   /* Offset to TCP/IP Id      */
s119IDLen  = c2d(substr(header,29,Ø2))   /* Length of TCP/IP Id.sec. */
s119IDNum  = c2d(substr(header,31,Ø2))   /* Number of TCP/IP Id      */
s119S1Off  = c2d(substr(header,33,Ø4))   /* Offset to TCP Conn close */
s119S1Len  = c2d(substr(header,37,Ø2))   /* Length of TCP Conn. close */
s119S1Num  = c2d(substr(header,39,Ø2))   /* Number of TCP Conn. close */
smfdate= left(Date('N',s119HDDate,'J'),11)
  Select
    when s119IDNum > Ø then call TCPIP_id
    otherwise nop
  End
return
TCPIP_id:
/*-------------------------------------------------------------------*/
/* Part 3: Common TCP/IP Identification Section - present in     */
/* every SMF type 119 record produced by the TCP/IP stack.       */
/* Its purpose is to identify the system and stack responsible   */
/* for producing the record.                                     */
/*-------------------------------------------------------------------*/
PARSE var rest
offØ1   =   s119IDOff- 43
s119TI_SYSName       = substr(rest,offØ1,8)    /* System name         */
s119TI_SysplexName   = substr(rest,offØ1+8,8)  /* Sysplex name        */
s119TI_Stack         = substr(rest,offØ1+16,8) /* Stack name          */
s119TI_ReleaseID     = substr(rest,offØ1+24,8) /* CS/39Ø release id.  */
s119TI_Comp          = substr(rest,offØ1+32,8) /* TCP/IP subcomponent */
s119TI_ASName        = substr(rest,offØ1+4Ø,8) /* ASID name of writer */
s119TI_UserID        = substr(rest,offØ1+48,8) /* User ID of security */
s119TI_ASID =     c2x(substr(rest,offØ1+58,2)) /* ASID of writer      */
s119TI_Reason =       substr(rest,offØ1+6Ø,1)  /* Reason for writing  */
 Select
  when s119TI_Comp = 'FTPC    ' then comp = "FTP Client"
  when s119TI_Comp = 'FTPS    ' then comp = "FTP Server"
  when s119TI_Comp = 'IP      ' then comp = "IP layer  "
  when s119TI_Comp = 'STACK   ' then comp = "Entire TCP/IP stack"
  when s119TI_Comp = 'TCP     ' then comp = "TCP layer     "
  when s119TI_Comp = 'TN3270C ' then comp = "TN327Ø Client "
  when s119TI_Comp = 'TN3270S ' then comp = "TN327Ø Server "
```

```
  when s119TI_Comp = 'UDP     ' then comp = "UDP layer     "
  otherwise                         comp = "... error ... "
 End
 s119TI_ReasonIntInc  = 'C0'x
 s119TI_ReasonInt     = '80'x
 s119TI_ReasonIEndInc = '60'x
 s119TI_ReasonIEnd    = '20'x
 s119TI_ReasonIShtInc = '50'x
 s119TI_ReasonISht    = '10'x
 s119TI_ReasonEvt     = '08'x
 if bitand(s119TI_Reason,s119TI_ReasonIntInc ) = s119TI_ReasonIntInc
then
 reason = 'C0 Interval statistics record, more records follow'
 if bitand(s119TI_Reason,s119TI_ReasonInt ) = s119TI_ReasonInt then
 reason = '80 Interval statistics record, last record in set'
 if bitand(s119TI_Reason,s119TI_ReasonIEndInc) = s119TI_ReasonIEndInc
then
 reason = '60 End-of-statistics record, more records follow'
 if bitand(s119TI_Reason,s119TI_ReasonIEnd ) = s119TI_ReasonIEnd then
 reason = '20 End-of-statistics record, last record in set'
 if bitand(s119TI_Reason,s119TI_ReasonIShtInc) = s119TI_ReasonIShtInc
then
 reason = '50 Shutdown starts record, more records follow'
 if bitand(s119TI_Reason,s119TI_ReasonISht ) = s119TI_ReasonISht then
 reason = '10 Shutdown starts record, last record in set'
 if bitand(s119TI_Reason,s119TI_ReasonEvt ) = s119TI_ReasonEvt then
 reason = '08 Event record'
/*-------------------------------------------------------------*/
/* Print System identification                                 */
/*-------------------------------------------------------------*/
 Select
     when inrec =  1 then do
     hds.1 = left("PART 1: SYSTEM ID:",44)
     hds.2 = left("System ID.  :",13) left(s119HDSID,8)
     hds.3 = left("System name :",13) left(s119TI_SYSName,8)
     hds.4 = left("Sysplex Name:",13) left(s119TI_SysplexName,8)
     hds.5 = left("Stack name  :",13) left(s119TI_Stack,8)
     hds.6 = left("Release ID  :",13) left(s119TI_ReleaseID,8)
     hds.7 = left(' ',1,' ')
    "EXECIO * DISKW CONN(STEM hds.)"
    "EXECIO * DISKW CONN(STEM hd.)"
     end
     otherwise nop
  End
  Select
    when s119S1Num > 0 then call TCP_close
    otherwise nop
  End
return
TCP_close: procedure EXPOSE      ,
```

```
port_count. port_inb. port_outb. ,
trip. conw. port_ins. port_outs. ,
s119S1Off   s119TI_UserID  s119TI_ASName s119TI_ASID,
rest inrec xmit rtt cw xm. Tot_xmit Tot_con   totinsg ,
totinb totoutb port_xmm.  port_time. Tot_elap totoutsg
/*-----------------------------------------------------------------*/
/* Part 4: TCP Connection Termination section                      */
/* Two notes:                                                      */
/* Since this information duplicates all of the information        */
/* contained in the TCP Connection Initiation record, it is        */
/* recommended that users only collect the TCP Connection          */
/* Termination record.                                             */
/* Because this record is generated for every single TCP connection*/
/* this can generate significant load on a server and rapidly fill */
/* the SMF datasets. Care should be exercised in its use.          */
/*-----------------------------------------------------------------*/
PARSE var rest
off   = s119S1Off  -43
s119_TTRName    = substr(rest,off,8)         /* Socket resource name  */
s119_TTConnID   = substr(rest,off+8,8)       /* TCP connection id     */
s119_TTSubTask  = c2d(substr(rest,off+16,4)) /* Subtask name: address */
                                             /* of MVS TCB for the    */
                                             /* task that owns this   */
                                             /* connection. Note that */
                                             /* this is not the       */
                                             /* subtask value specifie*/
                                             /* on an INITAPI call.   */
s119_TTSTime    = smf(c2d(substr(rest,off+20,4)))
                                             /* Connection start time */
s119_TTSDate    = ,                          /* Connection start date */
         substr(c2x(substr(rest,off+24,4)),3,5)
s119_TTETime=smf(c2d(substr(rest,off+28,4)))   /* Connection end time */
s119_TTEDate =,                               /* Connection end date */
                substr(c2x(substr(rest,off+32,4)),3,5)
condate= left(Date('N',s119_TTSDate,'J'),11)
enddate= left(Date('N',s119_TTEDate,'J'),11)
start    =     c2d(substr(rest,off+20,4))    /* Connection start time */
end      =     c2d(substr(rest,off+28,4))    /* Connection end time   */
elap     = cross(end,start)
Tot_elap = Tot_elap  + elap
elaps    = smf(elap)
s119_TTRIP =     substr(rest,off+36,16)      /* Remote IP addr.(IPv6 )*/
  s119_TTRIPRsvd     = substr(rest,off+36,12)  /*Unused if IPv4 format*/
    s119_TTRIPRsvd10 = substr(rest,off+36,10)     /* IPv6 reserved    */
    s119_TTRIPFmt1   = c2d(substr(rest,off+46,1))
                                             /*IPv4 address format - 1*/
    s119_TTRIPFmt2   = c2d(substr(rest,off+47,1))
                                             /*IPv4 address format - 2*/
  s119_TTRIP_IPv41 =  c2d(substr(rest,off+48,1))
                                             /*IPv4 piece 1 of address*/
```

```
  s119_TTRIP_IPv42 =  c2d(substr(rest,off+49,1))
                                          /*IPv4 piece 2 of address*/
  s119_TTRIP_IPv43 =  c2d(substr(rest,off+50,1))
                                          /*IPv4 piece 3 of address*/
  s119_TTRIP_IPv44 =  c2d(substr(rest,off+51,1))
                                          /*IPv4 piece 4 of address*/
fmt = s119_TTRIPFmt1||"."||s119_TTRIPFmt2
Rip =s119_TTRIP_IPv41||"."||s119_TTRIP_IPv42||"."||,
s119_TTRIP_IPv43||"."||s119_TTRIP_IPv44
s119_TTLIP =     substr(rest,off+52,16)  /* Local IP address (IPv6)   */
s119_TTLIPRsvd = substr(rest,off+52,12)  /* Unused if IPv4 addressing */
  s119_TTLIPRsvd10=substr(rest,off+52,10)     /* IPv6 reserved        */
  s119_TTLIPFmt1= c2d(substr(rest,off+62,1))  /* IPv4 address flagg1  */
  s119_TTLIPFmt2= c2d(substr(rest,off+63,1))  /* IPv4 address flagg2  */
s119_TTLIP_IPv1 = c2d(substr(rest,off+64,1)) /*IPv4 piece 1 of address*/
s119_TTLIP_IPv2 = c2d(substr(rest,off+65,1)) /*IPv4 piece 2 of address*/
s119_TTLIP_IPv3 = c2d(substr(rest,off+66,1)) /*IPv4 piece 3 of address*/
s119_TTLIP_IPv4 = c2d(substr(rest,off+67,1)) /*IPv4 piece 4 of address*/
Lip = s119_TTLIP_IPv1||"."||s119_TTLIP_IPv2||"."||,
s119_TTLIP_IPv2||"."||s119_TTLIP_IPv4
s119_TTRPort =c2d(substr(rest,off+68,2))      /* Remote port          */
s119_TTLPort =c2d(substr(rest,off+70,2))      /* Local port           */
s119_TTInBytes =c2d(substr(rest,off+72,8))    /* Inbound byte count   */
s119_TTOutBytes=c2d(substr(rest,off+80,8))    /* Outbound byte count  */
s119_TTSWS =c2d(substr(rest,off+88,4))        /* Send window size     */
s119_TTMSWS =c2d(substr(rest,off+92,4))       /* Max send window size */
s119_TTCWS  =c2d(substr(rest,off+96,4))       /*Congestion window size*/
s119_TTSMS=c2d(substr(rest,off+100,4))   /* Send segment size at close*/
s119_TTRTT=c2d(substr(rest,off+104,4))   /* Round trip time at close  */
s119_TTRVA=c2d(substr(rest,off+108,4))        /* RTT variance at close*/
s119_TTStatus =c2x(substr(rest,off+112,1))    /* Socket status        */
 Select
   when s119_TTStatus = '00'  then Socket = 'Server'
   otherwise                       Socket = 'Client'
 End
s119_TTTOS =c2x(substr(rest,off+113,1))       /* Type of service      */
s119_TTXRT=c2d(substr(rest,off+114,2))      /* Number of retransmits  */
s119_TTProf=    substr(rest,off+116,32)     /* Service profile name   */
s119_TTPol = substr(rest,off+148,32)        /* Service policy name    */
s119_TTInSeg=c2d(substr(rest,off+180,8))    /* Inbound segment count  */
s119_TTOutSeg=c2d(substr(rest,off+188,8))   /* Outbound segment count */
totinsg        = totinsg  + s119_TTInSeg
totoutsg       = totoutsg + s119_TTOutSeg
Tot_xmit       = Tot_xmit + s119_TTXRT
Tot_con        = Tot_con  + 1
trm            = s119_TTXRT
ins            = s119_TTInSeg
outs           = s119_TTOutSeg
lpn            = s119_TTLPort
inb            = s119_TTInBytes
```

```
outx           = s119_TTOutBytes
totinb         = totinb  + inb
totoutb        = totoutb + outx
port_count.lpn = port_count.lpn + 1
port_xmm.lpn   = port_xmm.lpn   + trm
port_inb.lpn   = port_inb.lpn   + inb
port_outb.lpn  = port_outb.lpn  + outx
port_ins.lpn   = port_ins.lpn   + ins
port_outs.lpn  = port_outs.lpn  + outs
port_time.lpn  = port_time.lpn  + elap
/*-----------------------------------------------------------------*/
/* Calculate:                                                      */
/* 1. Send window index                                            */
/* 2. Congestion window index                                      */
/* 3. Inbound byte rate (per second)                               */
/* 4. Outbound byte rate (per second)                              */
/* 5. Max traffic rate                                             */
/*-----------------------------------------------------------------*/
Select
   when (s119_TTSWS > Ø)  then do
   if s119_TTMSWS > Ø then ,
   senwindex = format(((s119_TTSWS/s119_TTMSWS)*1ØØ),4,2)
   end
   otherwise        senwindex = 'n.a'
End
Select
   when (s119_TTCWS > Ø)  then do
   if s119_TTMSWS > Ø then ,
   congindex = format(((s119_TTCWS/s119_TTMSWS)*1ØØ),4,2)
   end
   otherwise           congindex = 'n.a'
End
Select
   when (s119_TTInBytes > Ø)  then do
    if elap > Ø then insec = ,
    format((s119_TTInBytes/elap/1ØØ),7,2)
    end
   otherwise                    insec = Ø
End
Select
   when (s119_TTOutBytes > Ø)  then do
    if elap > Ø then outsec = ,
    format((s119_TTOutBytes/elap/1ØØ),7,2)
    end
   otherwise                     outsec = Ø
End
 Select
  when s119_TTRTT > Ø then ,
  maxrate = format(((s119_TTCWS/s119_TTRTT)/1.Ø24),8,Ø)
  otherwise  maxrate = "n.a"
```

```
 End
/* Print */
protd = Pxlate(s119_TTLPort)
rc.1 = left(enddate,12),              /* Connection end   time      */
       left(s119_TTETime,13),         /* Connection end   time      */
       left(elaps,12)       ,         /* Connection in hh:mm:ss:tt  */
       right(elap/100,8)    ,         /* Connection duration (sec.) */
       left(s119TI_ASName,8),         /* ASID name of writer        */
       left(s119TI_UserID,8),         /* User ID of security        */
       left(s119TI_ASID,4)  ,         /* ASID of writer             */
       left(s119_TTRName,8) ,         /* TCP socket name            */
       left(socket,7)       ,         /* Socket status              */
       left(s119_TTConnID,8),         /* Connection ID              */
       left(protd,26)       ,         /* Port description           */
       left(Lip,13)         ,         /* Local IP address           */
       right(s119_TTLPort,4),         /* Local port number          */
       left(Rip,13)         ,         /* Remote IP address          */
       right(s119_TTRPort,4),         /* Remote port number         */
       right(s119_TTRTT,3)  ,         /* Round Trip Time            */
       right(s119_TTRVA,4)  ,         /* Round Trip Time Variance   */
       right(MaxRate,8)     ,         /* Max Rate                   */
       right(s119_TTInBytes,5),       /* Inbound byte count         */
       right(insec,6)       ,         /* Inbound byte per sec.      */
       right(s119_TTOutBytes,5),      /* Outbound byte count        */
       right(outsec,6)       ,        /* Outbound byte per sec.     */
       right(s119_TTInSeg,6) ,        /* Inbound segment count      */
       right(s119_TTOutSeg,6),        /* Outbound segment count     */
       right(s119_TTSMS,6)   ,        /* Send Segment Size          */
       right(s119_TTXRT,3)   ,        /* Number of retransmission   */
       right(s119_TTSWS,8)   ,        /* Send Window Size           */
       right(s119_TTMSWS,7)  ,        /* Maximum Send Window Size   */
       right(s119_TTCWS,7)   ,        /* Congestion Window Size     */
       right(senwindex,7)    ,        /* Send Window index          */
       right(congindex,7)             /* Congestion Window index    */
  "EXECIO * DISKW CONN(STEM rc.)"
/*-------------------------------------------------------------------*/
/* Select records for PART 3, PART 4 & PART 5 of report:             */
/*        PART 3: Connections with retransmissions > 5               */
/*        PART 4: Connections with Round Trip Time > 400 ms          */
/*        PART 5: Connections with congestion window > 10 max        */
/*-------------------------------------------------------------------*/
 Select
   when s119_TTXRT > 5 then do
      xmit = xmit + 1
      xm.xmit = rc.1
     end
   otherwise nop
 End
Select
  when s119_TTRTT > 400 then do
```

```
  rtt = rtt +1
  trip.rtt = rc.1
  end
  otherwise nop
End
Select
  when (s119_TTMSWS  > Ø)  then do
   if (s119_TTCWS  > s119_TTMSWS*1Ø) then do
   cw = cw +1
   conw.cw = rc.1
  end; end
  otherwise nop
End
return
SMF: procedure
  /*-------------------------------------------------------------*/
 /* REXX - convert an SMF time to hh:mm:ss:hd format           */
  /*-------------------------------------------------------------*/
arg time
time1 = time % 1ØØ
 hh = time1 % 36ØØ;                    hh = right("Ø"||hh,2)
 mm = (time1 % 6Ø) - (hh * 6Ø);        mm = right("Ø"||mm,2)
 ss = time1 - (hh * 36ØØ) - (mm * 6Ø); ss = right("Ø"||ss,2)
 fr = time //  1ØØØ;                   fr = right("Ø"||fr,2)
rtime = hh||":"||mm||":"||ss||":"||fr
return rtime
CROSS: procedure
/*-------------------------------------------------------------*/
/*        Cover the midnight crossover                         */
/*-------------------------------------------------------------*/
arg endtime,startime
  select
   when endtime >= startime then nop
   otherwise  endtime = endtime + 8640ØØØ
  end
diftm =  endtime - startime
return diftm

Printline:
   /*-------------------------------------------------------------*/
  /*    Print each report line                                   */
   /*-------------------------------------------------------------*/
   PARSE arg lineout1
    "EXECIO 1 DISKW CONNS (STEM lineout)"
    if rc ¬= Ø then
      do
        say "printline RC =" RC
        exit rc
      end                                 /* end of printline */
 Return
```

```
Pxlate:
/*-------------------------------------------------------------*/
/* Port assignment lookup table                                */
/* Please note that this is only a sample list which must      */
/* be updated to reflect your installation's port assignment.  */
/*-------------------------------------------------------------*/
Parse Arg port
 Select
  when port =    7  then Des = "TCP/UDP  Echo             "
  when port =    9  then Des = "TCP/UDP  Discard          "
  when port =   19  then Des = "TCP/UDP  CharGen          "
  when port =   20  then Des = "TCP      FTP (data)       "
  when port =   21  then Des = "TCP      FTP (control)    "
  when port =   23  then Des = "TCP      Telnet           "
  when port =   25  then Des = "TCP      SMTP Server      "
  when port =   53  then Des = "TCP/UDP  DNS              "
  when port =   80  then Des = "TCP      HTTP             "
  when port =  111  then Des = "TCP/UDP  Portmap          "
  when port =  135  then Des = "UDP      NCS Loc.Broker   "
  when port =  161  then Des = "UDP      SNMP Agent       "
  when port =  162  then Des = "UDP      SNMP Query       "
  when port =  443  then Des = "TCP      HTTPS            "
  when port =  512  then Des = "TCP      Remote Exec      "
  when port =  514  then Des = "TCP      Remote Exec      "
  when port =  515  then Des = "TCP      LPD Server       "
  when port =  520  then Des = "UDP      OROUTED Serv     "
  when port =  580  then Des = "UDP      NCPROUTE Serv    "
  when port =  750  then Des = "TCP/UDP  Kerberos         "
  when port =  751  then Des = "TCP/UDP  Kerberos Admin   "
  when port = 1389  then Des = "TCP      LDAP             "
  when port = 1443  then Des = "TCP      HTTPS            "
  when port = 1933  then Des = "TCP      ILM MA Port      "
  when port = 1934  then Des = "TCP      ILM AA Port      "
  when port = 1933  then Des = "TCP      ILM MT Agent     "
  when port = 1934  then Des = "TCP      ILM LM Appl Agent"
  when port = 2809  then Des = "TCP      ORB port         "
  when port = 3000  then Des = "TCP      Prod  CICS Socket"
  when port = 3001  then Des = "TCP      Test1 CICS Socket"
  when port = 3002  then Des = "TCP      Test2 CICS Socket"
  when port = 4463  then Des = "TCP      DB2              "
  when port = 8801  then Des =" TCP      RMF/PM Java      "
  when port = 8880  then Des =" TCP      SOAP JMX Conn.   "
  when port = 9080  then Des =" TCP      HTTP port        "
   otherwise             Des =" fill in gap               "
    End
 Return  Des
```

Networking companies Attachmate and WRQ have finally merged and the new name for the company is AttachmateWRQ.

AttachmateWRQ products include DATABridge, e-Vantage, EXTRA!, INFOConnect, KEA!, NetWizard, Reflection, Synapta, and Verastream.

In the meantime, the WRQ part is shipping new versions of its Reflection host access software, and also enhancing its security offering, Reflection for Secure IT (aka F-Secure SSH). These are part of WRQ's vision of "universal host access", the ability to secure host connections and data while simplifying administration and deployment tasks.

Enhancements include integrated SSO for secure authentication, integrated SSH for secure simplified management of multiple servers, authentication of host access portlets for WebSphere, integrated SSH client capabilities across the Reflection product line including capabilities for reconfiguration of SSH Windows client sessions, group policy support for Reflection for Secure IT Windows client, automatic portlet generation capabilities for WebSphere, and Web-based management of Reflection FTP client configurations.

For further information contact:
URL: www.wrq.com/products/reflection/web.

* * *

NetManage has announced a new program that offers customers using Attachmate and WRQ products a migration path to NetManage products. Called NetManage Now!, the program offers migration plus maintenance and support.

NetManage products include RUMBA, OnWeb, and the NetManage Librados Adapters.

For further information contact:
URL: www.netmanage.com/pressroom/viewpress.asp?id=390.

* * *

NetManage has also announced the launch of its Librados Data Integration Plug-In. The Integration Plug-in will allow enterprise system functions and data, which are dynamically exposed in XML, to be mapped to a variety of data formats, including XML, database, flat file, and EDI. The J2EE standards-based plug-in connects back-end enterprise system functions, reducing implementation time for integration projects.

The Data Integration Plug-In is incorporated into the design-time configuration environment provided with each NetManage Librados JCA Plus Adapter. Exposed through the adapter via a GUI, the user is able to select business functions in the target enterprise application, transforming or translating source system data into any format, without the need for programming.

For further information contact:
URL: www.netmanage.com/pressroom/viewpress.asp?id=396

* * *