REXX Socket Programming

Author

Ursula Braun-Krahl
REXX/VSE Development, IBM Böblingen Laboratory
braunu@de.ibm.com

Interprocess communication within a computer or between computers is based on protocols. One concept for interprocess communication is the *socket*. It is the endpoint of a communication path.

The socket API is a low-level, general, and flexible application programming interface, very popular especially within the TCP/IP transport protocol. It allows to write communicating applications that receive data from a network and send data to a network.

The socket API is based on an *open/close/read/write* paradigm. Thus network communication is handled similar to reading from and writing to any other device.

All implementations of the socket programming interface are based on the original Berkeley Software Distribution (BSD) socket implementation with its roots in the UNIX environment.

TCP/IP Concepts

To understand socket programming, some basic TCP/IP concepts are listed here and explained shortly. For a more detailed explanation, one of the members of the rich set of TCP/IP books is recommended.

TCP/IP Protocols

A protocol is a set of rules or standards that two entities must follow to exchange and interpret messages. For communication via a network, TCP/IP defines a protocol stack consisting of different layers together with different protocols assigned to these layers. Famous protocols are:

IP - Internet Protocol

The IP layer provides the basic packet delivery services, but does not guarantee error-free arrival of IP packets in a determined order.

TCP - Transmission Control Protocol

TCP is a connection-oriented transport protocol that provides a reliable, full-duplex byte stream. It is used by the majority of TCP/IP applications.

UDP - User Datagram Protocol

UDP is a connectionless protocol that provides datagram services. It does not guarantee safe arrival of UDP datagrams. Such reliability features must be built into UDP applications.

IP Addresses

An IP address identifies the machine within a network. In the current version, it occupies 32 bits and is usually expressed externally in dotted decimal form, for instance "9.164.182.254".

Host name

In addition to the number form of IP Addresses, TCP/IP provides facilities to assign a symbolic name to an IP address, known as *host name*.

They are resolved in either some local host tables or via a special name server.

Ports A port number identifies a specific application on a given TCP/IP machine. It is a 16-bit integer ranging from 0 to 65534. A client application must know the port number of a server application to be able to contact it. Ports 0 to 1023 are reserved for well-known services. Ports 1024 to 5000 are used by TCP/IP for automatic assignments. Server applications should use port numbers above 5000.

Sockets The endpoint of a communication link between two application ports is uniquely identified by 3 components making up a *socket*

- protocol (TCP, UDP, IP)
- local IP-address
- local port

A *socket descriptor* or *socket number* is a 2-byte integer that acts as an index into a table of currently allocated sockets. Thus it represents the socket.

Socket Types

Three different socket types are defined:

Stream socket

connection-oriented (i.e. logical connection is established before data exchange), full-duplex, reliable, virtually unlimited size of byte streams, flow-control, TCP as default protocol, FTP as sample application

Datagram socket

connection-less (i.e. each datagram must contain the full set of addressing information for its delivery), unreliable, no flow-control, maximum size for a single datagram, UDP as default protocol, NFS as sample application

Raw socket

connection-less, unreliable, maximum size for messages, IP as default protocol, PING as sample application, not supported in REXX/VSE

Addressing Families

A socket identifies the address of a specific process at a specific computer using a specific transport protocol. The exact syntax of a socket address depends on the protocol being used, on its *addressing family* The only addressing family used in VSE is:

AF_INET

Addressing family Internet

Typical Socket Calls

A basic connection-oriented protocol like TCP consists of the following sequence of socket calls:

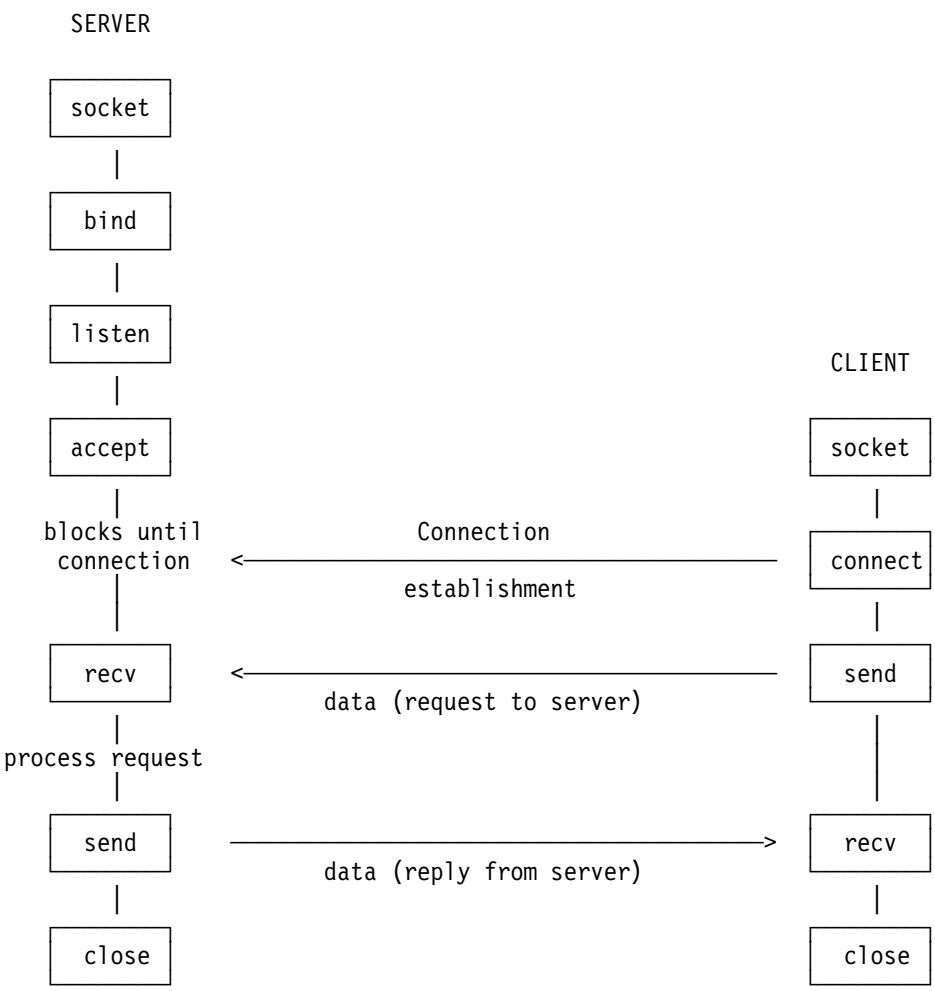


Figure 20. Connection-Oriented Protocol

A basic connection-less protocol like UDP consists of the following sequence of socket calls:

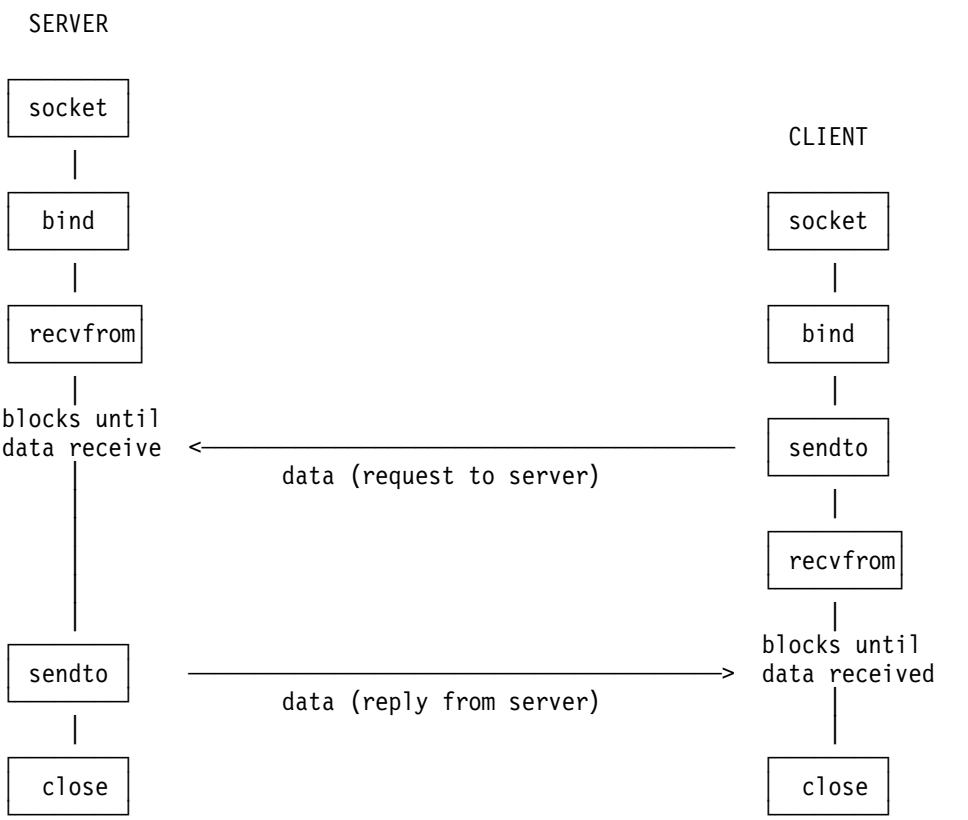
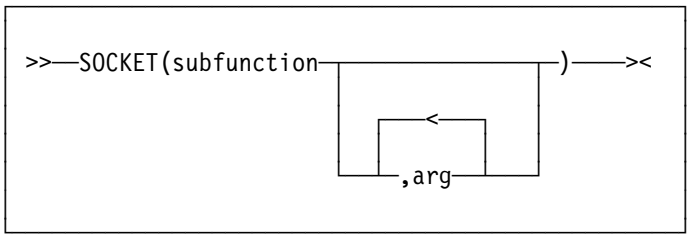


Figure 21. Connectionless Protocol

REXX Sockets Application Program Interface

With year-end 1999, REXX/VSE provided a REXX Socket API within PTF UQ37224 for VSE/ESA 2.3 and UQ37225 for VSE/ESA 2.4 (APAR PQ31258). It allows you to write socket applications in REXX/VSE for the TCP/IP environment. The REXX Socket Interface (i.e. a new function called "SOCKET") has the same syntax and semantic as the SOCKET function available for REXX/VM and REXX in OS/390.



subfunction
name of the socket call

arg
parameters for the socket call

A character string is returned from the SOCKET call that contains several values separated by blanks, so the string can be parsed using REXX. The first value in the string is usually the return code. If the return code is zero, the values following

the return code are returned by the socket subfunction. If the return code is not zero, the second value is the name of an error, and the rest of the string the corresponding error message.

Here are some examples:

```
SOCKET('GetHostId')    ==>  '0 9.4.3.2'
SOCKET('Recv',socket)  ==>  '1102 EWOULDBLOCK Problem on non-blocking socket'
```

The REXX/VSE SOCKET implementation uses the LE/VSE support to access the TCP/IP socket interface. Thus it maps the socket calls from the C programming language to REXX/VSE.

Tasks You Can Perform Using REXX Sockets

You can use REXX sockets to perform the following tasks:

1. Processing socket sets

   A socket set is a number of preallocated sockets available to a single application.

Table 2. REXX Socket Functions for Processing Socket Sets

Function	Purpose
Initialize	Defines a socket set
Terminate	Closes all the sockets in a socket set and releases the socket set
SocketSet	Gets the name of the active socket set
SocketSetList	Gets the names of all the available socket sets
SocketSetStatus	Gets the status of a socket set

2. Creating, connecting, changing, and closing sockets

   A socket is an endpoint for communication that can be named and addressed in a network.

Table 3. REXX Socket Functions for Creating, Connecting, Changing, and Closing Sockets

Function	Purpose
Socket	Creates a socket in the active socket set
Bind	Assigns a unique local name (network address) to a socket
Listen	Converts an active stream socket to a passive socket
Connect	Establishes a connection between two stream sockets
Accept	Accepts a connection from a client to a server
Shutdown	Shuts down a duplex connection
Close	Shuts down a socket

3. Exchanging data

   You can send and receive data on connected stream sockets and on datagram sockets.

Table 4. REXX Socket Functions for Exchanging Data

Function	Purpose
Read	Reads data on a connected socket
Write	Writes data on a connected socket
Recv	Receives data on a connected socket
Send	Sends data on a connected socket
RecvFrom	Receives data on a socket and gets the sender's address
SendTo	Sends data on a socket, and optionally specifies a destination address

4. Resolving names and other identifiers

   You can get information such as name, address, client identification, and host name. You can also resolve an Internet Protocol address (IP address) to a symbolic name of a symbolic name to an IP address.

Table 5. REXX Socket Functions for Resolving Names and Other Identifiers

Function	Purpose
GetHostId	Gets the IP address for the host processor
GetHostName	Gets the name of the host processor
GetSockName	Gets the local name to which a socket was bound
GetHostByAddr	Gets the host name for an IP address
GetHostByName	Gets the IP address for a host name
Resolve	Resolves the host name through a name server

5. Managing configurations, options, and modes

   You can obtain the version number of the REXX Sockets function package, get socket options, set socket options, and set the mode of operation.

Table 6. REXX Socket Functions for Managing Configurations, Options and Modes

Function	Purpose
Version	Gets the version and date of the REXX Sockets function package
Select	Monitors activity on selected sockets
GetSockOpt	Gets the status of options for a socket
SetSockOpt	Sets options for a socket
Fcntl	Sets or queries the mode of a socket
Ioctl	Controls the operating characteristics of a socket

Complete reference material for the new REXX/VSE Socket function is on the VSE/ESA home page:

http://www.ibm.com/s390s/vse/vsehtmls/vserxsoc.htm

REXX Samples Using the Socket Function

The following samples show how to make use of the REXX/VSE Socket function to establish and carry out (network) communication based on sockets. They demonstrate the simplicity of socket programming in REXX/VSE.

Two pairs of communicating REXX Socket programs are provided:

- SERVMIRR - CLIEMIRR : demo of basic socket concepts
- SERVSAMP - CLIESAMP : comprehensive demo of REXX socket programming

Source code for downloading as well as more samples will be available via the REXX/VSE home page:

http://www.ibm.com/s390/vse/rexxhome.htm

Here you'll also find counterparts to a REXX/VSE socket program that are written in Java for REXX for Windows NT/95. Thus they run on a different platform and communicate with the REXX/VSE socket program on a VSE/ESA host.

Even though the servers and clients shown here are coded in REXX/VSE, this does not mean that server and client have to run both on a VSE platform. A REXX/VSE socket program can communicate with any socket program written in any language supporting the socket API on any platform supporting TCP/IP, as long as the counterpart program uses the corresponding sequence of socket calls.

Sending and Receiving Data: Simple Server to Mirror Data

The mirror server receives a string from the client, reverses the order of the characters in the string and returns it to the client. After returning the mirrored string, communication is closed.

```
/*********************************************************************/
/***** SERVMIRR:                                                 ******/
/*********************************************************************/
/***** REXX program for a socket server procedure receiving     ******/
/***** data from a client, reversing the data, and sending      ******/
/***** the data back to the client.                             ******/
/*********************************************************************/

/***** initializations ************************************** ******/
rc = 0
trace off

/***** initialize socketset ********************************** ******/
fc = SOCKET('INITIALIZE','SERVMIRR')
parse var fc socket_rc .
if socket_rc ¬= 0
then do
  say 'INITIALIZE failed with return info ' fc
  exit 99
end
```

Figure 22 (Part 1 of 4). Server to Mirror Data

```
/***** create a TCP socket for client connection requests ***** ******/
fc = SOCKET('SOCKET','AF_INET','SOCK_STREAM','IPPROTO_TCP')
parse var fc socket_rc newsocketid
if socket_rc ¬= 0
then do
  say 'SOCKET failed with return info ' fc
  fc = SOCKET('TERMINATE')
  exit 99
end

/***** bind socket to well known port 1996 ******************** ******/
fc = SOCKET('BIND',newsocketid,'2 1996 9.164.155.114')
parse var fc bind_rc rest
if bind_rc ¬= 0
then do
  say 'BIND failed with return info ' fc
  fc = SOCKET('CLOSE',newsocketid)
  fc = SOCKET('TERMINATE')
  exit 99
end

/***** create a connection queue for a client ***************** ******/
fc = SOCKET('LISTEN',newsocketid,'10')
parse var fc listen_rc rest
if listen_rc ¬= 0
then do
  say 'LISTEN failed with return info ' fc
  fc = SOCKET('CLOSE',newsocketid)
  fc = SOCKET('TERMINATE')
  exit 99
end

/***** wait for a client to connect ************************** ******/
fc = SOCKET('ACCEPT',newsocketid)
parse var fc accept_rc rest
if accept_rc ¬= 0
then do
  say 'ACCEPT failed with return info ' fc
  fc = SOCKET('CLOSE',newsocketid)
  fc = SOCKET('TERMINATE')
  exit 99
end
parse var rest accepted_socket accept_socket_address
say "Client has established connection."
```

Figure 22 (Part 2 of 4). Server to Mirror Data

```
/***** we don't want any more clients, close request socket *** ******/
fc = SOCKET('CLOSE',newsocketid)
parse var fc close_rc rest
if close_rc ¬= 0
then do
  say 'CLOSE failed with return info ' fc
  exit 99
end

/***** read string from client ****************************** ******/
fc = SOCKET('READ',accepted_socket,'10000')
parse var fc read_rc num_read_bytes read_string
if read_rc ¬= 0
then do
  say 'READ failed with return info ' fc
  rc = 99
  signal SHUTDOWN_LABEL
end
say "String read from client: '" read_string "'"

/***** reverse string from client *************************** ******/
send_string = Reverse(read_string)

/***** send string back to client *************************** ******/
fc = SOCKET('SEND',accepted_socket,send_string,'')
parse var fc send_rc num_sent_bytes
if send_rc ¬= 0
then do
  say 'SEND failed with return info ' fc
  rc = 99
  signal SHUTDOWN_LABEL
end
/***** plausibility check *********************************** ******/
if num_read_bytes ¬= num_sent_bytes
then do
  say 'number of sent bytes does not match number of read bytes'
  rc = 99
  signal SHUTDOWN_LABEL
end
```

Figure 22 (Part 3 of 4). Server to Mirror Data

```
/***** close client socket ********************************** ******/
SHUTDOWN_LABEL:
fc = SOCKET('CLOSE',accepted_socket)
parse var fc close_rc rest
if close_rc ¬= 0
then do
  say 'CLOSE failed with return info ' fc
  fc = SOCKET('TERMINATE')
  exit 99
end

/***** terminate  socketset ********************************** ******/
fc = SOCKET('TERMINATE')
exit rc
```

Figure 22 (Part 4 of 4). Server to Mirror Data

A socket set must be initialized first before using any other REXX Socket function. Then a stream-type socket is created and bound to the well-known port number 1996 and the server IP address. Clients that want to connect to this server have to specify this port number and IP address.

With socket function LISTEN request, queues for possibly connecting clients are established; and afterwards the mirror server is prepared to accept its first client connection request. ACCEPT blocks the server until the first client starts connecting. It returns a new socket id to be used to communicate with the client. Connection of further clients may still be handled via further ACCEPT invocations using the original socket ID. Here it is not intended to support more clients. Thus the original socket is closed.

The client string is read, reversed and given back to the client. Afterwards, the server closes communication and terminates the whole socketset.

Sending and Receiving Data: Simple Client to Have Data Mirrored

A REXX client program able to talk to the previous mirror server begins on the next page:

```
/*********************************************************************/
/***** CLIEMIRR:                                                 ******/
/*********************************************************************/
/***** REXX program for a socket client procedure sending        ******/
/***** data to a server and receiving the reversed data from     ******/
/***** the server.                                               ******/
/*********************************************************************/

/***** initializations ************************************** ******/
rc = 0
trace off

/***** string to be reversed is given as parameter ************ ******/
arg read_string

/***** initialize socketset ********************************** ******/
fc = SOCKET('INITIALIZE','CLIEMIRR')
parse var fc socket_rc .
if socket_rc ¬= 0
then do
  say 'INITIALIZE failed with return info ' fc
  exit 99
end

/***** create a TCP socket *********************************** ******/
fc = SOCKET('SOCKET','AF_INET','STREAM','TCP')
parse var fc socket_rc newsocketid
if socket_rc ¬= 0
then do
  say 'SOCKET failed with return info ' fc
  fc = SOCKET('TERMINATE')
  exit 99
end

/***** connect new socket to the specified server ************* ******/
fc = SOCKET('CONNECT',newsocketid,'AF_INET 1996 9.164.155.114')
parse var fc connect_rc rest
if connect_rc ¬= 0
then do
  say 'CONNECT failed with return info ' fc
  rc = 99
  signal SHUTDOWN_LABEL
end
```

Figure 23 (Part 1 of 2). Client to Have Data Mirrored

```
/***** send string to the mirror server ********************** ******/
fc = SOCKET('SEND',newsocketid,read_string,'')
parse var fc send_rc num_sent_bytes
if send_rc ¬= 0
then do
  say 'SEND failed with return info ' fc
  rc = 99
  signal SHUTDOWN_LABEL
end

/***** plausibility check ********************************** ******/
if length(read_string) ¬= num_sent_bytes
then do
  say 'number of sent bytes does not match number of read bytes'
  rc = 99
  signal SHUTDOWN_LABEL
end

/***** receive answer from mirror server ********************* ******/
fc = SOCKET('READ',newsocketid,'10000')
parse var fc read_rc num_read_bytes received_string
if read_rc ¬= 0
then do
  say 'READ failed with return info ' fc
  rc = 99
  signal SHUTDOWN_LABEL
end
say "String '" read_string "' was mirrored to: '" received_string "'"

/***** close the socket ************************************ ******/
SHUTDOWN_LABEL:
fc = SOCKET('CLOSE',newsocketid)
parse var fc close_rc rest
if close_rc ¬= 0
then do
  say 'CLOSE failed with return info ' fc
  fc = SOCKET('TERMINATE')
  exit 99
end

/***** terminate  socketset ********************************** ******/
fc = SOCKET('TERMINATE')
exit rc
/*********************************************************************/
```

Figure 23 (Part 2 of 2). Client to Have Data Mirrored

After socket set initialization, the stream-type socket is created and connected to the well-known port and server address. Using the established connection, the given input string is sent to the server and the following READ blocks the client until the server has sent an answer. Then this specific socket communication is closed first, and afterwards the whole REXX-Socket communication setup is terminated.

Server for VSE Files and Console Commands

A more complex, but also more meaningful, REXX Socket sample follows. It is able to handle more than one client connected to the server at the same time. After receiving a retrieval command, it transfers library members, VSAM files, POWER queue entries, or console command responses back to its clients.

```
/*********************************************************************/
/*- SERVSAMP -- Example demonstrating the usage of REXX Sockets -----*/
/*********************************************************************/
/***** REXX program for a socket server procedure waiting for   ******/
/***** clients to connect, receiving one of the commands:       ******/
/*****    READLIB   -  retrieve a library member                ******/
/*****    READVSAM  -  retrieve a VSAM file                     ******/
/*****    READPOW   -  retrieve a POWER queue entry             ******/
/*****    console command  -  execute                           ******/
/***** and sending the resulting library member, VSAM file,     ******/
/***** POWER queue entry or console command response back to    ******/
/***** the client.                                              ******/
/*********************************************************************/

/***** initializations ************************************** ******/
/***** Port can be specified as argument. Default port is 5678. ******/
  trace o
  Parse Arg Port
  If Port = "" Then
    Port = 5678

/***** Open socket at well known port and wait for clients **** ******/
  SocketNr = ListenPort(Port)
  If SocketNr = -1 Then Do
    Call Socket 'Terminate'
    Exit 1
  End

/***** Close the socket when program is interrupted *********** ******/
  Signal On Halt
  Call Opermsg 'On'
  timeout = 15
  fc = ASSGN('STDOUT','SYSLOG')
  Say "Enter 'MSG " || SYSPID ||",DATA=HI' to exit program."
  fc = ASSGN('STDOUT','SYSLST')
```

Figure 24 (Part 1 of 14). Server for VSE Files and Console Commands

```
/***** Check for new events: ********************************** ******/
/*****   A new client may want to start communication with this ******/
/*****   server, or an existing client may want to sent data,   ******/
/*****   or the timeout interval has finished without any       ******/
/*****   special event.                                         ******/
  Do Forever
    sockevtlist = 'Read * Write Exception'
    parse value Socket('Select',sockevtlist,timeout) with rc socknum sellist
    If rc /= 0 Then
      Do
        Say "??? Select failed ???"
        Call Close SocketNr
        Return -1
      End
    parse upper var sellist . 'READ' orlist 'WRITE' . 'EXCEPTION' .
    If orlist /= '' Then Do
      event = 'SOCKET'
      parse var orlist orsocket .
      rest = 'READ' orsocket
    End
    Else event = 'TIME'
```

Figure 24 (Part 2 of 14). Server for VSE Files and Console Commands

```
    Select

      /* Accept connections from clients */
      When event = 'SOCKET' Then Do
        parse var rest keyword ts .

        /* Accept new connections from clients */
        if keyword = 'READ' & ts=SocketNr Then Do
          /* wait for client to connect and start handler    */
          Say "Waiting for client to connect."
          Parse Value Socket('Accept',SocketNr) ,
            with rc ClientSocket NetworkAddress
          Say "Client connected"
        End

        /* Receive Command */
        if keyword = 'READ' & ts/=SocketNr Then Do
          Command = ReceiveRequest(ts)
          If Command = 'QUIT' Then Do
            Call Close ts
            Say 'SERVSAMP: Disconnected socket' ts
          End
          Else Do
          End
        End

      End /* When */

      When event = 'TIME' Then Do
        Say "Timeout occurred"
      End

      /* Unknown event */
      Otherwise Nop
    End /* Select */
  End /* Do Forever */

  Halt:
  Call Close SocketNr
  Call Socket 'Terminate'
  Exit
```

Figure 24 (Part 3 of 14). Server for VSE Files and Console Commands

```
/*********************************************************/
/*                                                       */
/* Procedure: Close                                      */
/* Purpose:   Close the specified socket.                */
/* Arguments: Socket - active socket number              */
/* Returns:   nothing                                    */
/*                                                       */
/*********************************************************/
Close: Procedure
  Parse Arg SocketNr
  Call Socket 'Close',SocketNr
  Return

/*********************************************************/
/*                                                       */
/* Function:  ListenPort                                 */
/* Purpose:   Create a socket, bind it to a port and     */
/*            listen at the port for connecting clients.*/
/* Arguments: Port - port number                         */
/* Returns:   Socket number if successful, -1 otherwise */
/*                                                       */
/*********************************************************/
ListenPort: Procedure
  Parse Arg Port

  /* initialize socketset                               */
  parse value Socket('Initialize','SERVSAMP') ,
    with rc .
  If rc /= 0 Then
    Do
      Say "Unable to initialize socketset"
      Return -1
    End

  /* create a TCP socket                                */
  parse value Socket('Socket','AF_INET','Sock_stream','0') ,
    with rc SocketNr
  If rc /= 0 Then
    Do
      Say "Unable to create socket"
      Return -1
    End
```

Figure 24 (Part 4 of 14). Server for VSE Files and Console Commands

```
/* find out local IP address and bind socket to port  */
parse value Socket('GetHostId') with rc IpAddr
Host = "AF_INET" Port IpAddr
rc = Socket('Bind',SocketNr,Host)
if rc /= 0 Then
  Do
    Say "Unable to bind to port:" Port
    Call Close SocketNr
    Return -1
  End

/* listen at the port, allow 5 clients in queue       */
rc = Socket('Listen',SocketNr,5)
if rc /= 0 Then
  Do
    Say "Unable to listen at port:" Port
    Call Close SocketNr
    Return -1
  End

Return SocketNr
```

Figure 24 (Part 5 of 14). Server for VSE Files and Console Commands

```
/*******************************************************/
/*                                                     */
/* Function:  ReceiveRequest                           */
/* Purpose:   Wait for a command from the client and   */
/*            execute it. Return the identifier of the */
/*            command to the caller.                   */
/* Arguments: Socket - active socket number            */
/* Returns:   command identifier                       */
/*                                                     */
/*******************************************************/
ReceiveRequest: Procedure
  Parse Arg SocketNr

  /* wait for the command from the client              */
  parse value Socket('Recv',SocketNr,1024) ,
    with rc BytesRcvd CommandLine

  Say "Command line from client:" CommandLine

  CommandLine = Translate(CommandLine)
  If rc /= 0 Then
    CommandLine = "QUIT"
  Select;
  When Word(CommandLine,1) = "QUIT" Then
    Nop
  When Word(CommandLine,1) = "READLIB" Then
    Call ProcessLib SocketNr, Commandline
  When Word(CommandLine,1) = "READVSAM" Then
    Call ProcessVsam SocketNr, Commandline
  When Word(CommandLine,1) = "READPOW" Then
    Call ProcessPow  SocketNr, Commandline
  Otherwise Call ProcessCommand SocketNr, Commandline
  End; /* Select */

  /* send end of answer marker back to client */
  Call Socket 'Send',SocketNr,">>>End_of_transmission<<<"
  Return CommandLine
```

Figure 24 (Part 6 of 14). Server for VSE Files and Console Commands

```
/*******************************************************/
/*                                                     */
/* Procedure: Answer                                   */
/* Purpose:   Send one answer line back to the client  */
/*            and wait for acknowledgement from client.*/
/* Arguments: Socket - active socket number            */
/*            AnswerString - line to send to client    */
/* Returns:   nothing                                  */
/*                                                     */
/*******************************************************/
Answer: Procedure
  Parse Arg SocketNr, AnswerString
  Call Socket 'Send',SocketNr,AnswerString
  Call Socket 'Recv',SocketNr,256
  Return

/*******************************************************/
/*                                                     */
/* Procedure: AnswerQueue                              */
/* Purpose:   Send all lines contained in AnswerLines. */
/*            back to the client as the answer of the  */
/*            previous executed command.               */
/* Arguments: Socket - active socket number            */
/* Returns:   nothing                                  */
/*                                                     */
/*******************************************************/
AnswerQueue: Procedure Expose AnswerLines.
  Parse Arg SocketNr

  /* send answer lines until session queue is empty    */
  Do I = 1 to AnswerLines.0

    /* empty lines will be sent as space               */
    If AnswerLines.I = "" Then
      AnswerLines.I = " "

    Call Answer SocketNr, AnswerLines.I
  End
  Drop AnswerLines.
  Return
```

Figure 24 (Part 7 of 14). Server for VSE Files and Console Commands

```
/*******************************************************/
/*                                                     */
/* Procedure: ProcessCommand                           */
/* Purpose:   Process the console command that was     */
/*            received from the client and send back   */
/*            the result.                              */
/* Arguments: Socket - active socket number            */
/*            CommandLine - console command            */
/* Returns:   nothing                                  */
/*                                                     */
/*******************************************************/
ProcessCommand: Procedure
  Parse Arg SocketNr, CommandLine
  Call SYSVAR SYSPID
  carToken = 'SERVER' || SYSPID

  /* activate console session                         */
  ADDRESS CONSOLE 'ACTIVATE NAME' carToken 'PROFILE REXNORC'
  ADDRESS CONSOLE 'CART' carToken

  /* issue the command */
  ADDRESS CONSOLE CommandLine

  /* get the command result */
  fc = GETMSG('AnswerLines.','RESP',carToken,,15)

  /* deactivate console session                       */
  ADDRESS CONSOLE 'DEACTIVATE' carToken

  if AnswerLines.0 = 0 Then
   Do
     AnswerLines.0 = 1
     AnswerLines.1 = '??? no result ???'
   End

  Call AnswerQueue SocketNr
  Return
```

Figure 24 (Part 8 of 14). Server for VSE Files and Console Commands

```
/*******************************************************/
/*                                                     */
/* Procedure: ProcessLib                               */
/* Purpose:   Transfer the library member back to the   */
/*            client.                                  */
/* Arguments: Socket - active socket number            */
/*            CommandLine - READLIB request            */
/* Returns:   nothing                                  */
/*                                                     */
/*******************************************************/
ProcessLib: Procedure
  Parse Arg SocketNr, CommandLine

  Parse Var   CommandLine ,
        "READLIB " lib "." slib "." memname "." memtype .

  /* check for existence of the desired library member */
  Call REXXIPT 'libr_in.'
  Call OUTTRAP 'libr_out.',,'NOCONCAT'
  libr_in.0 = 2
  libr_in.1 = 'SEARCH ' || memname || '.' || memtype || ,
              ' SUBLIB=' || lib || '.' || slib
  libr_in.2 = '/*'
  ADDRESS LINK 'LIBR'
```

Figure 24 (Part 9 of 14). Server for VSE Files and Console Commands

```
If RC = 0
Then Do
  /* read the desired library member */
  libr_in.0 = 3
  libr_in.1 = 'ACCESS SUBLIB=' || lib || '.' || slib
  libr_in.2 = 'LIST ' memname || '.' || memtype
  libr_in.3 = '/*'
  ADDRESS LINK 'LIBR'
  If RC = 0
  Then Do
    Do I=1 to libr_out.0-7
      AnswerLines.I = value('libr_out.' || I+6)
    End
    AnswerLines.0 = libr_out.0-7
  End
  Else Do
    AnswerLines.0 = 1
    AnswerLines.1 = '??? Problem reading library member ???'
  End
End

Else Do
  /* desired library member has not been found */
  AnswerLines.0 = 1
  If RC = 2
  Then
    AnswerLines.1 = '??? Library member not found ???'
  Else
    AnswerLines.1 = '??? Problem accessing library member ???'
End

if AnswerLines.0 = 0 Then
 Do
   AnswerLines.0 = 1
   AnswerLines.1 = '??? no result ???'
 End

Call AnswerQueue SocketNr
Return
```

Figure 24 (Part 10 of 14). Server for VSE Files and Console Commands

```
/*********************************************************/
/*                                                       */
/* Procedure: ProcessVsam                                */
/* Purpose:   Transfer the VSAM file back to the client.*/
/* Arguments: Socket - active socket number              */
/*            CommandLine - READVSAM request             */
/* Returns:   nothing                                    */
/*                                                       */
/*********************************************************/
ProcessVsam: Procedure
  Parse Arg SocketNr, CommandLine
  Parse Var   CommandLine ,
        "READVSAM" filename catalog .
  If catalog = '' Then catalog = 'IJSYSCT'
  /* read the desired VSAM file */
  ADDRESS JCL
    "// DLBL PRINTFL,'" || filename || "',,VSAM,CAT=" || catalog
    "/*"
  Call REXXIPT 'idcams_in.'
  Call OUTTRAP 'idcams_out.',,'NOCONCAT'
  idcams_in.0 = 1
  idcams_in.1 = 'PRINT INFILE (PRINTFL) CHARACTER'
  ADDRESS LINK 'IDCAMS MARGINS(1 80)'
  If RC = 0
  Then Do
    Do I=1 to idcams_out.0-11
      AnswerLines.I = value('idcams_out.' || I+7)
    End
    AnswerLines.0 = idcams_out.0-11
  End
  Else Do
    AnswerLines.0 = 1
    AnswerLines.1 = '??? Problem reading VSAM file ???'
  End

  if AnswerLines.0 = 0 Then
   Do
     AnswerLines.0 = 1
     AnswerLines.1 = '??? no result ???'
   End

  Call AnswerQueue SocketNr
  Return
```

Figure 24 (Part 11 of 14). Server for VSE Files and Console Commands

```
/*********************************************************/
/*                                                       */
/* Procedure: ProcessPow                                 */
/* Purpose:   Transfer POWER queue entry back to the     */
/*            client.                                    */
/* Arguments: Socket - active socket number              */
/*            CommandLine - READPOW request              */
/* Returns:   nothing                                    */
/*                                                       */
/*********************************************************/
ProcessPow: Procedure
  Parse Arg SocketNr, CommandLine
  Parse Var   CommandLine ,
        "READPOW" queue entryname entrynum entryclass .
  if queue /= 'RDR' & queue /= 'LST' & queue /= 'PUN'
  Then Do
    AnswerLines.0 = 1
    AnswerLines.1 = '??? queue specification invalid ???'
    Call AnswerQueue SocketNr
    Return
  End
```

Figure 24 (Part 12 of 14). Server for VSE Files and Console Commands

```
Call OUTTRAP 'power_out.',,'NOCONCAT'
/* determine user of POWER queue entry */
ADDRESS POWER
'PDISPLAY ' || queue || ',' || entryname

Do I=2 To power_out.0
  If entrynum = ''
  Then Do
    entrynum = word(power_out.I,3)
  End
  If entryclass = ''
  Then Do
    entryclass = word(power_out.I,6)
  End
  If word(power_out.I,2) = entryname & ,
     word(power_out.I,3) = entrynum & ,
     word(power_out.I,6) = entryclass
  Then Do
    Parse Var power_out.I . 'FROM=(' from_user ')' .
    Parse Var power_out.I . 'TO=(' to_user ')' .
    Select
      When from_user /= '' Then setuid_user = from_user
      When to_user /= '' Then setuid_user = to_user
      Otherwise Do
        setuid_user = 'REXX'
        fc = SENDCMD('PALTER' queue || ',' || entryname || ,
                     ',' || entrynum || ',USER=' || setuid_user ,
                    )
      End
    End /* Select */
    Leave
  End
End
```

Figure 24 (Part 13 of 14). Server for VSE Files and Console Commands

```
/* read the desired POWER queue entry */
'SETUID' setuid_user
ADDRESS POWER 'GETQE' queue 'JOBNAME' entryname ,
                            'JOBNUM' entrynum ,
                            'CLASS' entryclass ,
                            'STEM AnswerLines.'
If RC = 0 & power_out.0 = 0
Then Do
  Nop
End
Else Do
  AnswerLines.0 = 1
  AnswerLines.1 = '??? Problem reading POWER queue entry ???'
End

if AnswerLines.0 = 0 Then
 Do
   AnswerLines.0 = 1
   AnswerLines.1 = '??? no result ???'
 End

Call AnswerQueue SocketNr
Return
```

Figure 24 (Part 14 of 14). Server for VSE Files and Console Commands

This server runs until an operator interrupts it via the console operator communication exit:

```
MSG <partition_id>,DATA=HI
```

Using the Socket SELECT function, the server determines whether a new client is starting communication or an existing client is issuing a new server request. Incoming requests that do not start with one of the keywords *QUIT*, *READLIB*, *READVSAM*, or *READPOW* are treated as console commands and given to the console router using the REXX console command environment.

To transfer a library member, the server checks first via LIBR SEARCH if the desired library member exists. If available, it moves its contents into a REXX stem variable and sends these lines to the client. To transfer a VSAM file, the server invokes IDCAMS PRINT to retrieve its data into a REXX stem variable, whose content is then sent to the client.

For POWER queue entries, the server determines existence and ownership of a requested entry using a PDISPLAY command. For entries without FROM- and TO-user assigned an artificial TO-user, REXX is defined to be authorized to access the POWER queue entry. Afterwards, the entry is retrieved via ADDRESS POWER GETQE into a REXX stem variable to be sent.

Client for VSE Files and Console Commands

A corresponding client for the previous REXX Socket sample is given here. It sends commands to the server requesting transfer of library members, VSAM files, POWER queue entries, or console command responses. For demonstration purposes, the received response is displayed at the console.

```
/*********************************************************************/
/*- CLIESAMP -- Example demonstrating the usage of REXX Sockets -----*/
/*********************************************************************/
/***** REXX program for a socket client procedure sending one   ******/
/***** of the commands                                          ******/
/*****    READLIB   -  retrieve a library member                ******/
/*****    READVSAM  -  retrieve a VSAM file                     ******/
/*****    READPOW   -  retrieve a POWER queue entry             ******/
/*****    console command  -  execute                           ******/
/***** to the server, and writing the data returned from the    ******/
/***** server to SYSLOG.                                        ******/
/*********************************************************************/

/***** initializations ************************************** ******/
/***** Server must be specified as argument.                  ******/
/***** Optionally a port can be specified in addition separated ******/
/***** by a colon. Default port is 5678                       ******/
  trace o
  Parse Arg Server
  CALL ASSGN 'STDIN','SYSLOG'              /* Input stream: SYSLOG  */
  CALL ASSGN 'STDOUT','SYSLOG'             /* Output stream: SYSLOG */

  /* check command line arguments, server is required */
  If Server = "" Then
   Do
     Say "Usage: CLIESAMP Servername"
     Say "  Servername may contain a port number separated with a colon."
     Exit 1
  End
```

Figure 25 (Part 1 of 4). Client for VSE Files and Console Commands

```
/* Connect to remote control server */
Socket = Connect(Server)
If socket = -1 Then Do
  Call Socket 'Terminate'
  Exit 1
End

/* loop until QUIT command was entered */
Do Until Command = "QUIT"
  Say "Please enter:"
  Say "     console command"
  Say "     'READLIB' lib.slib.memname.memtyp"
  Say "     'READVSAM' vsam.name catalog"
  Say "     'READPOW' RDR|LST|PUN name number class"
  Say "or   'QUIT'"
  Parse Pull CommandLine
  Parse Upper Var CommandLine Command Option
  If Length(Command) > 0 Then
    Call SendCommand Socket, CommandLine
End

/* Close connection to server */
Call Close Socket
Call Socket 'Terminate'
Exit
```

Figure 25 (Part 2 of 4). Client for VSE Files and Console Commands

```
/*******************************************************/
/*                                                     */
/* Function:  Connect                                  */
/* Purpose:   Create a socket and connect it to server. */
/* Arguments: Server - server name, may contain port no.*/
/* Returns:   Socket number if successful, -1 otherwise */
/*                                                     */
/*******************************************************/
Connect: Procedure
  Parse Arg Server

  /* if the servername has a port address specified     */
  /* then use this one, otherwise use the default port  */
  /* for the remote control server (5678)               */
  Parse Var Server Server ":" Port
  If Port = "" Then
    Port = 5678

  /* initialize a socketset                             */
  parse value Socket('Initialize','clielib') ,
    with rc .
  If rc /= 0 Then
    Do
      Say "Unable to initialize socketset"
      Return -1
    End

  /* create a TCP socket                                */
  parse value Socket('Socket','2','1','0') ,
    with rc SocketNr
  If rc /= 0 Then
    Do
      Say "Unable to create socket"
      Return -1
    End

  /* connect the new socket to the specified server     */
  Host = "AF_INET" Port Server
  rc = Socket('Connect',SocketNr,Host)
  if Words(rc) > 1 Then
    Do
      Say "Unable to connect to server:" Server
      Call Close SocketNr
      Return -1
    End

  Return SocketNr
```

Figure 25 (Part 3 of 4). Client for VSE Files and Console Commands

```
/*******************************************************/
/*                                                     */
/* Function:  SendCommand                              */
/* Purpose:   Send a command via the specified socket  */
/*            and display the full response from server */
/* Arguments: SocketNr - active socket number          */
/*            Command - command string                 */
/* Returns:   nothing                                  */
/*                                                     */
/*******************************************************/
SendCommand: Procedure
  Parse Arg SocketNr, Command

  /* send the command to the remote control server     */
  Call Socket 'Send', SocketNr, Command

  Do Forever
    parse value Socket('Recv',SocketNr,1024) ,
      with rc BytesRcvd RcvData

    /* error or end of response encountered            */
    If rc /= 0 | ,
       RcvData = ">>>End_of_transmission<<<" Then
      Leave

    /* display response and send acknowledge to server  */
    Say RcvData
    Call Socket 'Send', SocketNr, "OK!"
  End

  Say "----- end of output from command:" Command "-----"
  Return

/*******************************************************/
/*                                                     */
/* Function:  Close                                    */
/* Purpose:   Close the specified socket               */
/* Arguments: SocketNr - active socket number          */
/* Returns:   nothing                                  */
/*                                                     */
/*******************************************************/
Close: Procedure
  Parse Arg SocketNr
  Call Socket 'Close', SocketNr

  Return
/*********************************************************************/
```

Figure 25 (Part 4 of 4). Client for VSE Files and Console Commands

This client runs until command *QUIT* is entered at the console. All other console input is sent to the server, where it is either interpreted as some kind of file retrieval command (*READLIB*, *READVSAM*, or *READPOW*) or as console command.

Additional Information

Besides the complete reference material for the REXX/VSE Socket function on the VSE/ESA home page:

http://www.ibm.com/s390/vse/vsehtmls/vserxsoc.htm

you can find more information in:

- *TCP/IP for VSE/ESA User's Guide*, SC33-6601-01
- *MVS TCP/IP Sockets*, GG24-2561-00

  Even though the book is intended for MVS programmers, it contains lots of general information on socket programming valid also for VSE/ESA.
- TCP/IP for VSE/ESA home page
  http://www.ibm.com/s390/vse/vsehtmls/tcphome.htm
- REXX/VSE home page
  http://www.ibm.com/s390/vse/rexx/rexxhome.htm

If you have any questions or comments about this article or about REXX/VSE, please feel free to contact me using email at **braunu@de.ibm.com**.