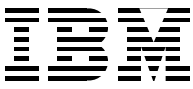
TCP/IP and REXX Programming with REXX/SOCKETS

SHARE, San Francisco, California,
February 28th – March 5th, 1993
and
SHARE Europe, Hamburg, Germany,
April 19th – 23rd, 1993
Session 2.7G

Arthur J. Ecock and **Rainer F. Hauser**

City University of New York
ECKCU@CUNYVM
eckcu@cunyvm.cuny.edu



Zurich Research Laboratory
RFH@ZURLVM1
rfh@zurich.ibm.com

February, March and **April 1993**

TCP/IP Overview

The Transmission Control Protocol/Internet Protocol (TCP/IP) is a communication facility based on the internet technology that has resulted from research funded by the Defense Advanced Projects Research Agency (DARPA).

An **internet** is a logical collection of networks supported by gateways, routers, bridges, hosts, and various layers of protocols. An internet permits different physical networks to function as a single, large virtual network, and permits dissimilar computers to communicate with each other.

A **host** is a computer, connected to a network, which provides an access point to that network. A **port** is an end point for communication between applications, generally referring to a logical connection.

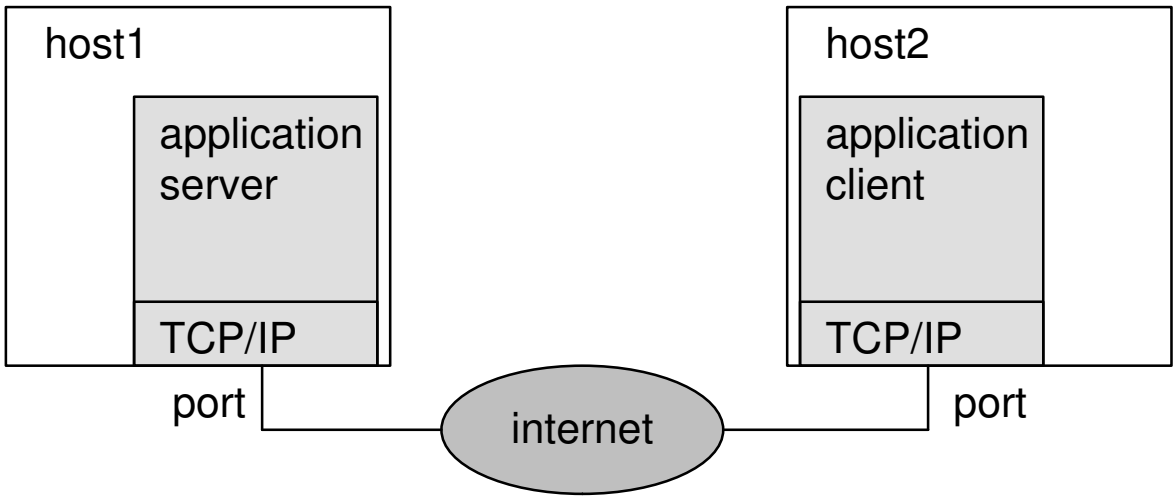
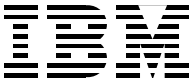


Figure 1: TCP/IP Communication Overview



TCP/IP Protocols and Layers

The TCP/IP layered architecture consists of many protocols:

- Network Layer:
  Token Ring, Ethernet, X.25, and others
- Internetwork Layer:
  Internet Protocol (IP), Internet Control Message Protocol (ICMP), Address Resolution Protocol (ARP)
- Transport Layer:
  Transmission Control Protocol (TCP), User Datagram Protocol (UDP)
- Application Layer:
  Telnet, File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP), Kerberos Authentication System, Remote Printing (LPR/LPD), X Window System, Socket Interfaces, and others.

TCP provides a reliable vehicle for delivering packets between hosts on an internet. **UDP** provides an unreliable mode of communication between source and destination hosts.

Socket Interfaces allow programmers to write their own applications to supplement those supplied by TCP/IP. Sockets permit full–duplex transmission (transmission in both directions simultaneously). Reading and writing to a socket appears very similar to performing I/O to a file or any other network device.

TCP/IP Connections

Note: The following is an example of connection–oriented use of TCP/IP. The connectionless mode is not discussed.

A TCP **connection** is a set of two **character** queues (one for each direction) which represent the connection over the network.

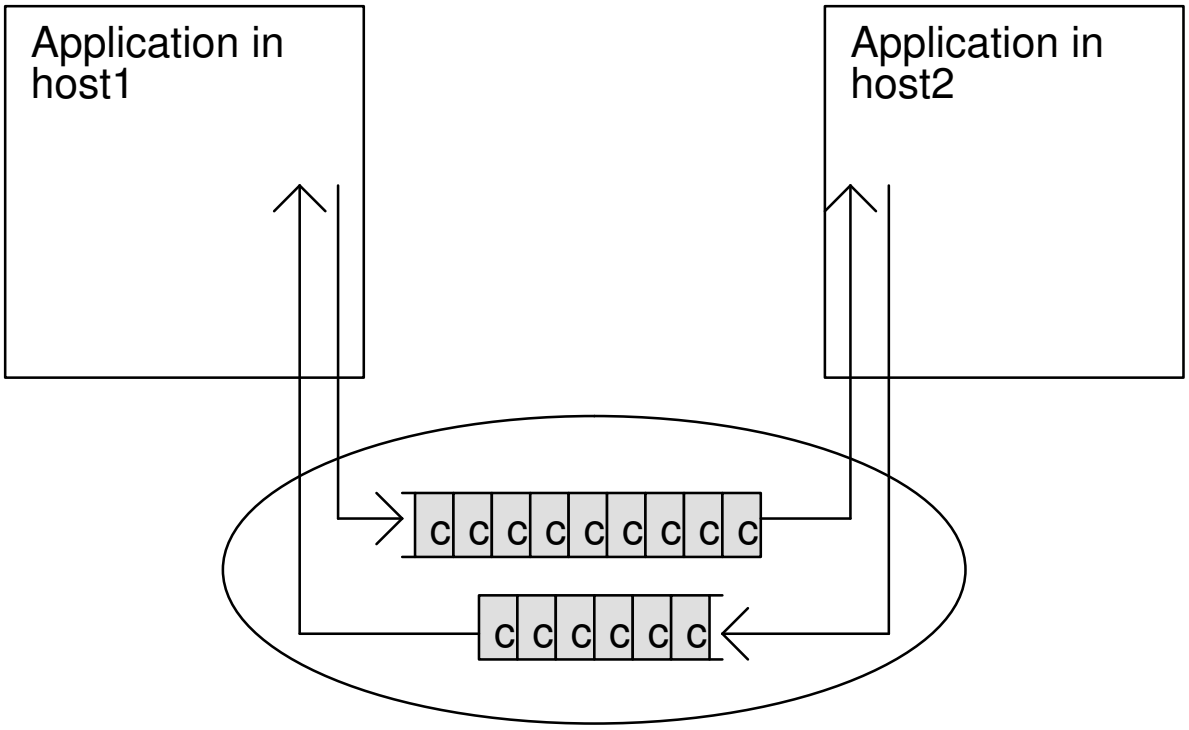
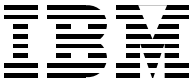


Figure 2: Character Queues for a TCP Connection

There are functions to add characters to a queue on the sender's side (Send, Write) and to remove characters from the queue on the receiver's side (Recv, Read) of an established connection. The socket interface is a convenient interface to issue these and other functions.



Socket Functions

The design of the socket interface is closely related to the programming language C.

```
int socket(int domain, int type, int protocol);
...
int s;
...
s = socket(AF_INET, SOCK_STREAM, 0);
```

There are many socket functions, but the following are of general interest:
socket() – gets a socket number to read from or write to
bind() – associates a socket with a port number
listen() – listens to connect requests on the socket
select() – waits for activity on a socket
connect() – requests a connection
accept() – accepts a connect request
send() or write() – writes data to the socket
recv() or read() – reads data from the socket
close() – returns socket number
givesocket() and takesocket() – transfer a socket

Addressing

When a client tries to connect to a server, the **internet address** and the **port number** must be known. The internet address identifies the host, and the port number identifies the application. Internet addresses are usually structured such as 128.228.1.2 (for host CUNYVM) or 9.4.3.2 (for host ZURLVM1).

Socket Protocol Overview

A typical TCP Socket session between a client and a server:

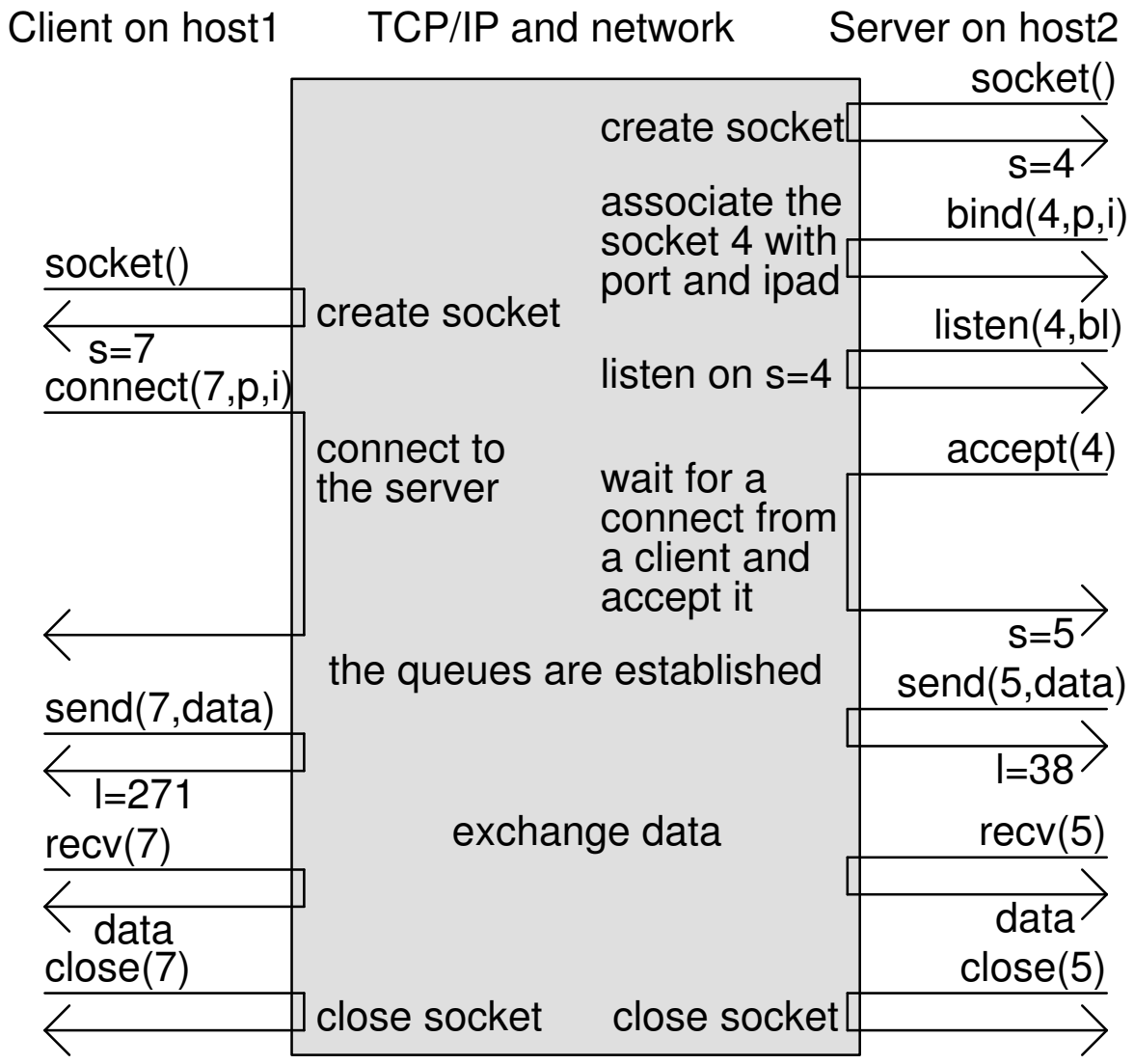


Figure 3: Typical TCP Socket Session

REXX/SOCKETS Overview

REXX/SOCKETS is a REXX interface to the TCP/IP Socket calls for CMS users.

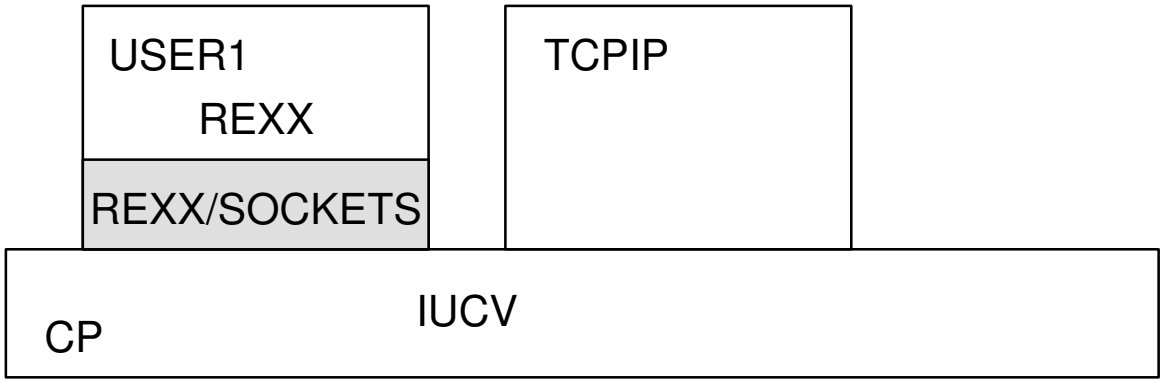


Figure 4: REXX/SOCKETS Overview

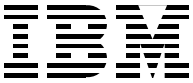
REXX/SOCKETS provides the REXX function SOCKET:

```
result = SOCKET(subfunction,arg1,...,argn)
```

Supported subfunctions are:
Initialize, Terminate, SocketSet, SocketSetStatus, ...
Socket, Bind, Listen, Connect, Accept, Shutdown, Close, ...
GiveSocket, TakeSocket, GetClientId, ...
GetHostId, GetHostName, GetHostByAddr, GetHostByName, ...
Read, Recv, RecvFrom, Write, Send, SendTo, ...
GetSockOpt, SetSockOpt, Ioctl, ...

Note: REXX/SOCKETS supports the non–blocking mode (and the Select socket function) through REXX/WAIT.

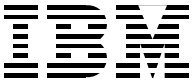
The result consists of a return code and additional data depending on which subfunction was called.



REXX/SOCKETS Concepts

The language definition and concepts of REXX guided the design of REXX/SOCKETS:

1. When a REXX program using REXX/SOCKETS terminates, the status of REXX/SOCKETS is kept. In other words, when a REXX program terminates, sockets are not closed and the characters in the queues of a connection remain.
2. The TCP/IP socket interface is not the only protocol which can be supported by REXX. Therefore, names such as CONNECT and ACCEPT should be avoided as names for REXX functions.
3. The arguments to the REXX function SOCKET and its result string are always character strings. The strings '128.288.1.2' and '9.4.3.2', for example, are valid internet addresses and can directly be used in subfunctions such as Bind.
4. The REXX function SOCKET does not have side effects. For example, no REXX variables (such as RC) are modified, and the result string contains all information available separated by blanks for easy parsing.
5. Waiting for one event within a given list of expected events is such an important function, that it must be provided directly by REXX and not by REXX/SOCKETS. Therefore, REXX/SOCKETS supports waiting (the Select socket function) through the central WAIT function in REXX/WAIT.



REXX/WAIT Overview

REXX/WAIT (when loaded as a nucleus extension) provides a set of additional REXX functions useful for REXX programs using the REXX interface to the TCP/IP socket calls:

```
result = WAIT(event1 [args1],...,eventn [argsn])
result = SETVALUE(event [args])
result = QUERYVALUE(event [args])
result = RESETVALUE(event)

ebcdicstring = AC2EC(asciistring)
asciistring = EC2AC(ebcdicstring)
type = CTYPE([type])
table = CTABLE(type,[option],[start],[end],[table])
```

The first four functions return in the result string a return code and other data depending on the function and the event name.

The SETVALUE, QUERYVALUE and RESETVALUE functions control the default arguments for waiting with the WAIT function which allows to wait for an event in a given set of possible events:

```
result = SETVALUE('SOCKET 5 NON-BLOCKING')
result = WAIT('CONS','SOCKET READ 5 WRITE 7 8')
result = WAIT('TIME 10MIN','SOCKET READ 5')
```

The AC2EC and EC2AC functions convert character strings from ASCII to EBCDIC encoding and vice versa. The translation table can be set with the CTABLE function. The CTYPE function allows to determine whether the machine on which the REXX program runs uses ASCII or EBCDIC encoding.

REXX/WAIT Low–Level Interface

Through the low–level interface provided by REXX/WAIT, other programs (such as REXX/SOCKETS) can export an event name (such as SOCKET) to become available for the WAIT, SETVALUE, QUERYVALUE and RESETVALUE functions.

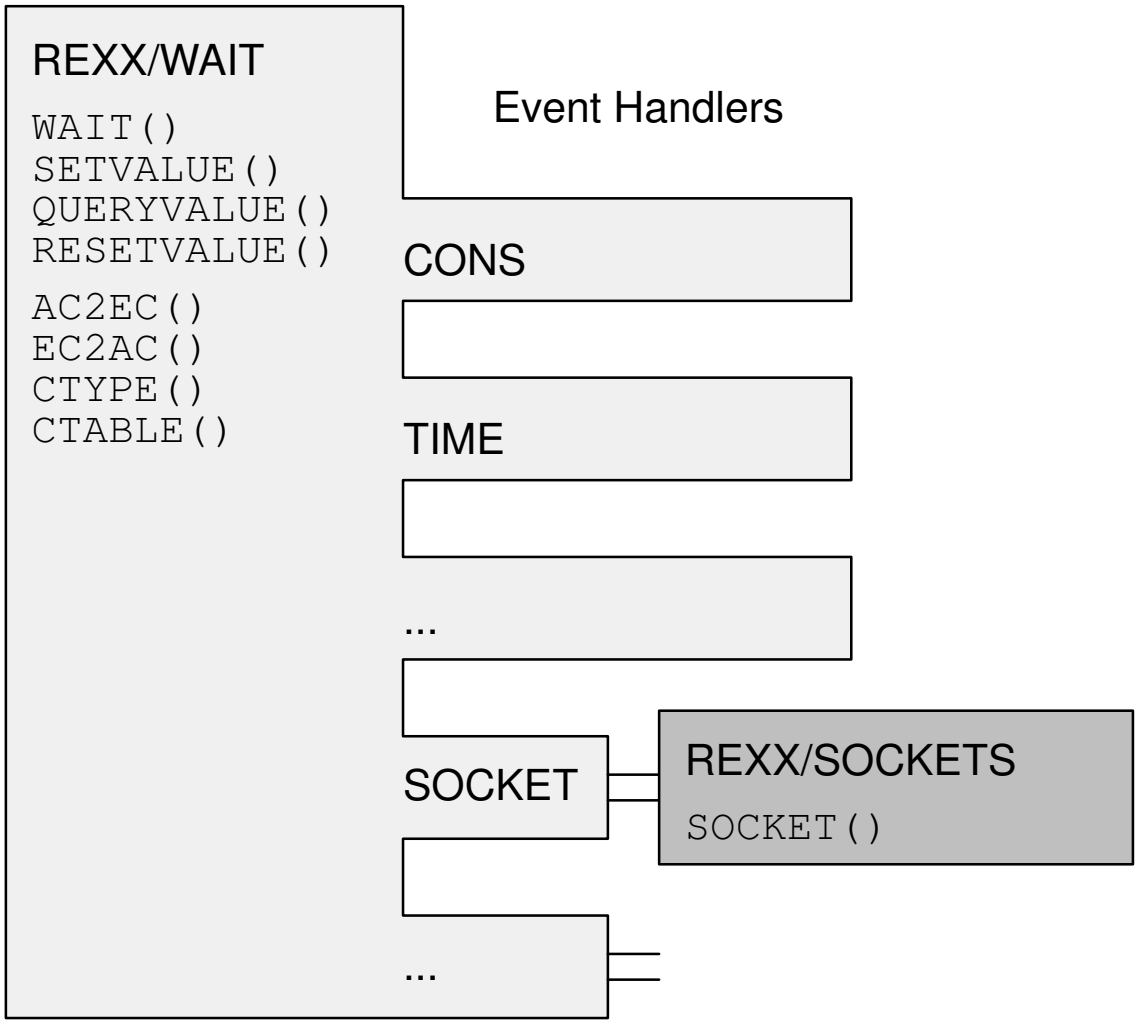
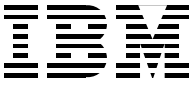


Figure 5: REXX/WAIT Interfaces



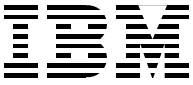
REXX/SOCKETS & REXX/WAIT Examples

The following are valid REXX statements and their possible results when REXX/SOCKETS and REXX/WAIT are available:

```
Socket('Initialize','myId')       == '0 myId 40 TCPIP'
Socket('Socket')                  == '0 1'
Socket('Connect',1,'AF_INET 1234 128.228.1.2') == '0'
SetValue('Socket 1 Non-Blocking') == '0 BLOCKING'
Socket('Recv',1) == '35 EWOULDBLOCK Operation would block'
Wait('Socket Read 1 2 3','Cons')  == '0 READ 1'
Socket('Recv',1) == '0 25 Here are twentyfive bytes'
Socket('Send',1,'We send twentyfour bytes') == '0 24'
Socket('ShutDown',1,'Both')       == '0'
Socket('Close',1)                 == '0'
Socket('SocketSet')               == '0 myId'
Socket('Terminate')               == '0 myId'
```

The subfunctions accepting internet addresses in their input parameters also allow names in the form 'CUNYVM' when it makes sense. As output in the result string, internet addresses are always in the form '128.228.1.2'.

```
Socket('Connect',1,'AF_INET 99 128.228.1.2') == '0'
Socket('Connect',1,'AF_INET 99 CUNYVM') == '0'
Socket('Connect',1,'AF_INET 99 CUNYVM.CUNY.EDU') == '0'
Socket('Accept',2) == '0 3 AF_INET 5678 9.4.3.2'
```

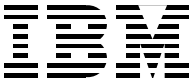


REXX/SOCKETS Server Sample Part 1

The server sample program accepts connect requests from clients and receives data:

```
/* RCVSAMPL -- REXX/SOCKETS server example */
parse arg .
parse value SOCKET('Initialize','Example') with src .
if src<>0 then exit src
parse value SOCKET('GetHostId') with src IpAddress .
parse value SOCKET('Socket') with src Socket .
call SOCKET 'BIND',Socket,'AF_INET 1993' IpAddress
call SOCKET 'LISTEN',Socket,10
call SETVALUE 'SOCKET' Socket 'NON-BLOCKING'
call SETVALUE 'SOCKET READ' Socket
do forever
  parse value WAIT('SOCKET','CONS') with wrc Type Descriptor
  say 'Event='Type '('Descriptor')'
  select
    when Type='SOCKET' then do
      call ProcessTcpip Descriptor
    end
    when Type='CONS' then do
      if Descriptor='EXIT' then leave
    end
  end
end
call SOCKET 'Close',Socket
call SOCKET 'Terminate'
exit 0
```

The actual processing of the Socket events is done in the subroutine `ProcessTcpip`.

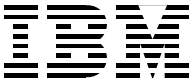


REXX/SOCKETS Server Sample Part 2

The subroutine `ProcessTcpip` which processes TCP/IP events:

```
ProcessTcpip:
  procedure expose Socket Counter.
  selector = arg(1)
  parse var selector Type SocketId
  if Socket=SocketId then do
    parse value SOCKET('Accept',Socket) with src SocketId Peer
    say 'TCP/IP Connection='Peer
    call SETVALUE 'SOCKET' SocketId 'NON-BLOCKING'
    parse value QUERYVALUE('SOCKET DEFAULTS') with wrc EList
    call SETVALUE 'SOCKET' EList SocketId
    Counter.SocketId = 0
  end
  else do
    parse value SOCKET('Recv',SocketId) with src msglen Message
    if Message<>'' then do
      Counter.SocketId = Counter.SocketId + length(Message)
      say Message
    end
    else do
      call SOCKET 'Close',SocketId
      say Counter.SocketId 'bytes received on socket' SocketId
    end
  end
return
```

Depending on the socketid, the subroutine either accepts the connect request or receives data. If the message is empty, the connection has been closed by the peer.



REXX/SOCKETS Client Sample

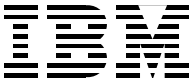
The client sample program connects to the server (on the same host) and sends data:

```
/* SNDSAMPL -- REXX/SOCKETS client example */
parse arg Count Message
parse value SOCKET('Initialize','Example') with src .
if src<>0 then exit src
parse value SOCKET('GetHostId') with src IpAddress .
parse value SOCKET('Socket') with src Socket .
call SOCKET 'Connect',Socket,'AF_INET 1993' IpAddress
do i=1 to Count by 1
  parse value SOCKET('Send',Socket,Message) with src msglen .
  say 'Sent' msglen 'bytes with rc='src
  if src<>0 then leave
end
call SOCKET 'Close',Socket
call SOCKET 'Terminate'
```

Comments

The server program sets all sockets to non–blocking mode and waits on events using REXX/WAIT. The client program however uses the blocking mode and therefore gets blocked in the connect and send subfunctions.

In order to keep the sample programs small, both programs do not include the testing necessary for real application programs. Since the first token of the result string is the return code, its value should always be checked.



REXX/SOCKETS History and Status

4/91	REXX/WAIT available on IBM's VMTOOLS disk
8/91	RXSOCKET available on BITNET's LISTSERV
12/91	REXX/SOCK available on IBM's VMTOOLS disk
3/92	Agreement between Arty Ecock and Rainer Hauser to combine RXSOCKET and REXX/SOCK
11/92	Syntax for REXX/SOCKETS complete: **`RXSOCKET + REXX/SOCK = REXX/SOCKETS`**
2/93	REXX/SOCKETS available on BITNET's LISTSERV and on IBM's VMTOOLS disk

Future

We hope that REXX/SOCKETS and REXX/WAIT together will be included in the next release of IBM's TCP/IP product for VM. There is also interest in making them available for MVS as well.

REXX/SOCKETS and REXX/WAIT have both been designed to be portable to REXX on other platforms. When available on platforms using ASCII encoding, carefully written REXX programs should continue to run unmodified.