

RPG Does TCP/IP

(Socket Programming in RPG IV)



Presented by

Scott Klement

<http://www.scottklement.com>

© 2007-2012, Scott Klement

“There are 10 types of people in the world.
Those who understand binary, and those who don’t.”

Objectives Of This Session



- Understand how to write TCP/IP Client Programs in RPG
- Understand how to write TCP/IP Server Programs in RPG
- Find any needed documentation for TCP/IP programming

This session will not cover the use of TCP/IP tools, such as the IBM i FTP client, PING or TRACERT commands. It's purely about building your own TCP/IP programs from the ground up.

Sample Scenarios using TCP/IP



- A custom-designed “thick-client” program. (Windows GUI and RPG back-end)
- Any place you might have written custom communications (such as ICF files) in the past can be replaced with a pure TCP/IP solution.
- Server-to-server interfacing. Real-time communication between two back-end servers (even if they're on different platforms and in different programming languages.)
- You may want to write your own tools that use standard Internet protocols such as Telnet, FTP, HTTP, E-mail, etc. You can write your own TCP/IP tools from the ground up in RPG. -- or invent your own!
- Direct communication with a sign, scale, printer, barcode scanner, etc.

3

TCP/IP is a Suite of Protocols



There are four *main* network protocols in the TCP/IP suite. The name “TCP/IP” is taken from the two most popular of these protocols.

HTTP, FTP, SMTP, TN5250, Custom	
TCP	UDP
ICMP, IP	
Ethernet, PPP, SLIP, etc.	

- **IP - Internet Protocol.** A low-level protocol that defines how a datagram is routed across multiple networks. (Not used directly by applications.)
- **ICMP – Internet Control Message Protocol.** A low-level protocol to provide diagnostic/error messages about IP. (Not used directly by applications.)
- **UDP – User Datagram Protocol.** Used by applications that want to create and send datagrams directly. Faster than TCP, but the operating system does less work for you. You must write code to split data into packets , verify receipt, verify correct sequence.
- **TCP – Transmission Control Protocol.** Operating system does most of the work for you. TCP breaks data into packets, verifies that they're received (resending if needed), and reassembles them in correct sequence on the remote side.

4

The Socket APIs



TCP/IP programs are written using a suite of Unix-type APIs called “Socket APIs”.

- **The End Point:** A socket is the “end point” of TCP/IP communications, much like a telephone receiver is the “end point” of a telephone conversation.
- **Communicate between your Program and IBM i:** Your program uses the socket to tell IBM i what work needs to be done:
 - What protocol you want to speak
 - Where (address / port) to send the data to
 - ... and the data to send to the other system.
- **A socket is to a network what a file is to a hard drive:** When you want to write data to disk, you don't have to break your data up into disk blocks, or tell the computer where to move the read/write head. Same thing with sockets, you just write data to the socket, let TCP work out the details. 5

TCP Sockets Work Like a Telephone



Client

- Look up the telephone number and extension number.
(Look up the IP address and port)
- Pick up the telephone receiver.
(Open a new socket)
- Dial the number
(Connect the socket)
- Talk.
(Send/receive data over the socket)
- Hang up.
(Close the socket)

Server

- Create the telephone (sigh)
(Open a new socket.)
- Decide which telephone extension to wait for.
(Bind the socket to a port.)
- Turn the ringer on.
(Tell socket to listen for connect)
- Wait for a call, then lift the receiver.
(Accept a new connection)
- Talk *(send/receive)*
- Hang up. *(close socket)*

The GETTIME Program



To demonstrate the concepts in this presentation, I've developed some simple example programs:

GETTIME – client program

- Accept an IP address or Domain name as a parameter.
- Connect to port 8123 on the server (specified in first parm)
- Send the "SEND TIME" command.
- Receive back the time stamp.
- Use DSPLY to display it on the screen.

SHOWTIME – server program

- Wait for a connection on port 8123
- Wait for a command from the client (only supported command is "SEND TIME")
- Send the current time from the system clock.
- Disconnect.

When I call GETTIME, it will produce results like this:

```
CALL GETTIME PARM('server6.scottklement.com')  
DSPLY  Server's time is: 2007-04-03-12.19.47.692000
```

7

Objective 1



HOW TO WRITE A CLIENT PROGRAM

8

Binary vs. Dotted Addresses



BINARY IP ADDRESS

Computer uses a 4-byte binary number (10U 0 in RPG), such as **x'C0A864C8'**

DOTTED IP ADDRESS

People print each byte as a separate number, separated with a "dot".

Example: 192.168.100.200

These represent the same address – just different ways of writing it. When the user gives you a "dotted" IP address, you need a way to convert it to binary format.

You do that with the `inet_addr()` API....

9

The `inet_addr()` API

Convert a "dotted" address to binary



The `inet_addr()` API converts a dotted address to a binary IP address.

```
D inet_addr      PR          10U 0 ExtProc('inet_addr')
D char_addr     *          *   value options(*string)
```

I have a copy book called `SOCKET_H` that contains all of the prototypes, named constants, and data structures used with the socket APIs. You can download it from my Web site at:

<http://www.scottklement.com/presentations/>

```
/copy socket_h

D dotted        s          15a
D ip_addr       s          10u 0

    dotted = '192.168.100.200';
    ip_addr = inet_addr(%trim(dotted));

// ip_addr is now 3232261320
```

Get IP Address for Domain Name



NAMES NOT NUMBERS!!

Instead of an IP address, the user might give you a name like:

www.google.com -or- isociety.common.org

In order for your system to translate names to numbers, it has to be told where to find a DNS server. You can specify an IP address for a DNS server on the following command:

```
CHGTCPDMN HOSTNAME('mycomputer')
           DMNNAME('example.com')
           HOSTSCHPTY(*LOCAL)
           INTNETADR('x.x.x.x')
```

You can run the DNS server on your IBM i box, but you don't have to. IBM i is quite happy to use a server running on Windows, Unix, etc. You can even use one provided by your ISP. Just put the correct IP address in the INTNETADR parm above.

11

Gethostbyname() API

Look up ("resolve") the IP address for a host name



gethostbyname() definition from SOCKET_H:

```
D gethostbyname PR * extProc('gethostbyname')
D HostName * value options(*string)

D hostent DS Based(p_hostent)
D h_name *
D h_aliases *
D h_addrtype 5I 0
D h_length 5I 0
D h_addrlist *
D p_h_addr S * Based(h_addrlist)
D h_addr S 10U 0 Based(p_h_addr)
```

Using gethostbyname() in your RPG program:

```
/copy socket_h
D host s 100A inz('isociety.common.org')
D ip_addr s 10u 0
. . .
p_hostent = gethostbyname(%trim(host));
if (p_hostent = *null);
errMsg = 'Host not found!';
else;
ip_addr = h_addr;
endif;
```

Performs lookup in BOTH
host table and DNS.

12

Handle Both IP Address and DNS



```
C      *ENTRY      PLIST
C      PARM          host
/free

    ip_addr = inet_addr(%trim(host));

    if (ip_addr = INADDR_NONE);
        p_hostent = gethostbyname(%trim(host));
        if (p_hostent = *null);
            errMsg = 'Host not found!'
        else;
            ip_addr = h_addr;
        endif;
    endif;
```

- First try inet_addr() to see if it's a valid IP address.
- If not valid, inet_addr() will return INADDR_NONE
- Then try gethostbyname().

Ports



Think about this:

- An IP address gets you to the right computer (actually, network interface!)
- How do you get to the right application (program) within that computer?
- Port numbers are sent in each packet to distinguish each program from the others.
- Servers usually use a “well-known” port number so clients know who to connect to. It's always the same for that particular server.
- Kinda like a telephone extension number. (dial 1-800-KLEMENT, then you hear "if you know your party's extension, please dial it now...")

FTP is always 21.

TELNET is always 23.

SMTP (email) is always 25.

HTTP is always 80

Ports above 4000 are used for "custom programming"

...etc...

The SHOWADDR (time server) example program will run on port 8123.

The socket() API

Pick Up the Phone / Create Socket



Once you know who you're connecting to, you need a socket to work with.

From SOCKET_H:

```
D socket          PR          10I 0 ExtProc('socket')
D AddrFamily      10I 0 Value
D SocketType      10I 0 Value
D Protocol        10I 0 Value
```

Sockets can be used with many different types of networks (IPv4, IPv6, IPX/SPX, UNIX domain). You have to tell the system what type you want.

- AddrFamily = address family (protocol family), i.e. **which protocol suite to use.**
 - AF_INET means TCP/IP (IPv4)
 - AF_INET6 means TCP/IP (IPv6)
- SocketType = Type of socket
 - SOCK_STREAM = stream socket (TCP)
 - SOCK_DGRAM = datagram socket (UDP)
- Protocol = Which protocol within the family.
 - IPPROTO_IP = Use TCP or UDP based on the SocketType parameter.

15

socket() API Sample Code



The socket() API returns a "socket descriptor", or -1 if an error occurs.

SOCKET DESCRIPTOR:

Because you can have many simultaneous sockets, and they don't have a name (like the filenames you'd put on an F-spec) you need a way to keep track of each socket you open.

The socket() API returns a number that IBM i uses internally to keep it straight from other connections.

You must save that number into a variable, and pass it to subsequent APIs to tell them which socket to operate on.

```
/copy socket_h

D mySock          s          10i 0
. . .
mySock = socket( AF_INET: SOCK_STREAM: IPPROTO_IP );
if (mySock = -1);
    // handle error
endif;
```

16

The connect() API

Dial Phone / Connect Socket



A socket address data structure (IPv4 version shown) stores info about the address and port to connect to. From SOCKET_H:

```
D sockaddr_in      DS                based(p_sockaddr)
D   sin_family    5I 0
D   sin_port      5U 0
D   sin_addr      10U 0
D   sin_zero      8A
```

- *Sin_Family* identifies which "address family" (or protocol suite = TCP/IP)
- *Sin_Zero* should always be hex zeroes.

```
D connect          PR                10I 0 ExtProc('connect')
D   Sock_Desc      10I 0 VALUE
D   p_SockAddr     *   VALUE
D   AddressLen     10I 0 VALUE
```

- *Sock_Desc* = the descriptor (the number returned by the socket() API)
- *p_SockAddr* = Address (%ADDR) of the socket address data structure.
- *AddressLen* = The size (%SIZE) of the socket address in bytes.

Returns 0 if connected successfully, or -1 if an error occurs.

17

connect() API Sample Code



In your RPG program:

```
/copy socket_h
D connto          ds                likeds(sockaddr_in)
D rc              s                10i 0

connto = *allx'00';
connto.sin_family = AF_INET; // Type of address
connto.sin_addr   = IP_Addr; // Result of DNS lookup
connto.sin_port   = 8123     // port number

rc = connect( mySock: %addr(connto): %size(connto) );
if (rc = -1);
    // error has occurred
endif;
```

Most errors in a TCP/IP client program occur during this API, because it's the first place where bytes are sent/received from the remote computer – so if there's something wrong with the connection, this is the first place you'll notice it!

18

The send() and recv() APIs

Talk / Hold a Conversation



```
D Send          PR          10I 0 ExtProc('send')
D  Sock_Desc    10I 0 Value
D  p_Buffer     *    Value
D  BufferLen    10I 0 Value
D  Flags       10I 0 Value
```

```
D Recv          PR          10I 0 ExtProc('recv')
D  Sock_Desc    10I 0 Value
D  p_Buffer     *    Value
D  BufferLen    10I 0 Value
D  Flags       10I 0 Value
```

- **Sock_Desc** = descriptor (value returned by socket() API)
- **p_buffer** = address of variable to send/receive.
- **BufferLen** = length of data to send, or size of variable to receive data into.
- **Flags** = Almost never used. Pass 0 for this parameter.

When receiving: How much data depends on how much is immediately available – NOT like a record. Variable won't always be filled, or match the exact size that was written on the other end.

Translating Data



Since most network applications communicate in ASCII, you'll need to be able to convert ASCII to EBCDIC and vice-versa.

```
D QDCXLATE      PR          ExtPgm('QDCXLATE')
D  len          5p 0 const
D  data         32767A options(*varsize)
D  table        10a constv
```

QDCXLATE is a system API that can translate data according to a table. IBM provides many tables, but for simple applications, I typically use these:

- QTCPASC = translate EBCDIC to ASCII
- QTCPEBC = translate ASCII to EBCDIC

QDCXLATE is a simple way to convert data, suitable for simple applications. The iconv() API is a better solution, but much more complicated, so I use QDCXLATE in my examples.

Sample Code to Send Data



This is code from a custom application that synchronizes time between two servers. After connecting, the client sends a command to the server that says 'SEND TIME'.

```
D cmd          s          20a
D len          s          10i 0
D CRLF         c          x'0d25`
. . .
  cmd = 'SEND TIME' + CRLF;
  QDCXLATE( %len(%trimr(cmd)): cmd: 'QTCPASC');

  len = send( mySock: %addr(cmd): %len(%trimr(cmd)): 0);
  if (len < %len(%trimr(cmd)));
    errMsg = 'Error during send.';
  endif;
```

There's nothing special about the string "SEND TIME", it could be any data I wanted to send to the server program. In this case, the server program waits for the words "SEND TIME" and when it receives them, it sends it's current time stamp.

21

Sample Code to Receive Data



The time stamp the server sends is always 26 bytes long, but depending on the network speed, it may not all arrive in one recv() call. I use a loop to add the data together til I have all 26 bytes.

```
D tempvar      s          26a
D response     s          26a  varying
. . .
  response = '';
  dou %len(response) = 26;

  len = recv( mySock: %addr(tempvar): %size(tempvar): 0);
  if (len = -1);
    errMsg = 'Error during recv.';
  endif;

  response = response + %subst(tempvar:1:len);
enddo;

len = %len(response);
tempvar = response;
QDCXLATE( len : tempvar: 'QTCPEBC' );
response = %subst(tempvar: 1: len);

dsply ('Server's time is: ' + response);
```

The close() API

Hang Up the Phone



```
/if not defined(CLOSE_PROTOTYPE)
D Close      PR          10I 0 ExtProc('close')
D  Sock_Desc  s          10I 0 Value
/define CLOSE_PROTOTYPE
/endif
```

- **Sock_Desc** = descriptor (value returned by socket() API)

Returns 0 when successful, -1 upon failure.

Sockets will not be closed automatically, even if you end your program with *INLR = *ON. You must remember to close them with this API!

```
callp close(mySock);
```

Tip: Sockets cannot be re-used. Once an error occurs, you must close the socket, and create a new one to try again.

23

Error Handling w/errno



Unix-type APIs report errors by setting an “error number”. The `__errno()` procedure can be used to retrieve a pointer to an error number. This error number is used with all C APIs and Unix-type APIs, not only sockets.

I put error handling stuff in the ERRNO_H member. Like SOCKET_H, you can download it from my Web site at <http://www.scottklement.com/presentations/>

```
D sys_errno      PR          * ExtProc('__errno')
D errno          s          10i 0 based(ptr)
. . .
mySock = socket(AF_INET: SOCK_STREAM: IPPROTO_IP);
if (mySock = -1);
    ptr = sys_errno();
    errMsg = 'Error number ' + %char(errno) + ' in socket() API';
endif;
```

The error number can be converted to a message using the `strerror()` API. For example:

```
rc = connect(mySock: %addr(connto): %size(connto));
if (rc = -1);
    ptr = sys_errno();
    errMsg = %str( strerror(errno) );
endif;
```

24

Error Constants



There are named constants that correspond to each possible value of “errno”. Here’s a partial list (from ERRNO_H). These are the error codes listed in the IBM manuals.

```
* Address already in use.
D EADDRINUSE      C          3420
* Address not available.
D EADDRNOTAVAIL  C          3421
* The type of socket is not supported in this address family.
D EAFNOSUPPORT   C          3422
* Operation already in progress.
D EALREADY       C          3423
* Connection ended abnormally.
D ECONNABORTED   C          3424
* A remote host refused an attempted connect operation
D ECONNREFUSED   C          3425
* A connection with a remote socket was reset by that socket.
D ECONNRESET     C          3426
* Operation requires destination address.
D EDESTADDRREQ   C          3427
* A remote host is not available.
D EHOSTDOWN      C          3428
```

25

Error Numbers Used in Program



In this example, a different message is supplied for some errors to make them clearer to the user. It illustrates how you might use an error constant in your program.

```
repeat = *off;
dou not repeat;

rc = connect( mySock: %addr(connto): %size(connto));
if (rc = -1);
  ptr = sys_errno();
  select;
  when errno = EINTR;
    repeat = *on;
  when errno = ECONNREFUSED;
    errMsg = 'No server program running on remote computer.';
  when errno = EHOSTDOWN;
    errMsg = 'Remote computer is down';
  other;
    errMsg = %str( strerror(errno) );
  ends!;
  exsr LogError;
endif;

enddo;
```

26



HOW TO WRITE A SERVER PROGRAM

27

The Easy Way to Write a Server



Due to time constraints, I'll only describe the "easy way" to create a TCP server in this talk.

Rather than doing all the work yourself, use INETD.

You configure it:

- Port server runs on (using service table)
- Type of socket (tcp or udp)
- Name of your program.

INETD will:

1. Sit and wait for a client to connect.
2. Create a new socket for the new connection. (It will always be #0)
3. Submit your program to batch (with the socket already connected) to talk to the client.

28

Setting Up Your Service



For time "TIMEDEMO" server (the one that returns the current time stamp for syncing the clock) to work, I need to register it with inetd. To do that, I have to follow these steps:

- Add my new custom service to the system's service table.

```
ADDSRVTBLE SERVICE('timedemo') PORT(8123) PROTOCOL('tcp')
```

- Edit inetd's configuration file to tell it which services to listen on, and which programs to submit for each service.

```
EDTF '/QIBM/USERDATA/OS400/INETD/inetd.conf'
```

Unfortunately, IBM does not provide a GUI tool (that I know of) for configuring INETD. Plus, the documentation is a bit sketchy. However, it's the same as the inetd programs that are ubiquitous on Unix systems, so the manual pages for Unix explain the setup:

Here's a link to the one for FreeBSD (open source Unix for the PC):

<http://www.freebsd.org/cgi/man.cgi?query=inetd.conf&format=html>

29

Service Table



Every system has a table (file) that lets you cross reference well-known services to port numbers.

- Not distributed across the Internet like DNS.
- More like the HOSTS table – just a file on each computer.

**WRKSRVTBLE
command:**

```
Work with Service Table Entries
System: S10561BA
Type options, press Enter.
 1=Add 4=Remove 5=Display

Opt Service          Port Protocol
-----
 1  rmtjournal         3777  tcp
 2  rmtjournal         3777  udp
 3  routed             520   udp
 4  sealm3             1025  tcp
 5  server3            54321 tcp
 6  sftp               115   tcp
 7  sftp               115   udp
 8  smtp              25    tcp
 9  smtp              25    udp
10  snmp              161   tcp
11  snmp              161   udp
More...

Parameters for options 1 and 4 or command
==>
F3=Exit  F4=Prompt  F5=Refresh  F6=Print list  F9=Retrieve  F12=Cancel
F17=Top  F18=Bottom
```

30

Configuring INETD



```
# Basic services
#echo          stream tcp nowait QTCP *INTERNAL
#discard      stream tcp nowait QTCP *INTERNAL
#chargen      stream tcp nowait QTCP *INTERNAL
#daytime      stream tcp nowait QTCP *INTERNAL
#time         stream tcp nowait QTCP *INTERNAL
#echo         dgram  udp  wait  QTCP *INTERNAL
#discard      dgram  udp  wait  QTCP *INTERNAL
#chargen      dgram  udp  wait  QTCP *INTERNAL
#daytime      dgram  udp  wait  QTCP *INTERNAL
#time         dgram  udp  wait  QTCP *INTERNAL
#
timedemo      stream tcp nowait QUSER /QSYS.LIB/MYLIB.LIB/SHOWTIME.PGM
```

- Lines that begin with # are comments.
- Timedemo is the service name for port 8123
- Socket type is "stream tcp"
- INETD will not wait for my program to complete after submitting it.
- My program will run with the authority of the QUSER user profile.
- My program is called SHOWTIME in library MYLIB

**Use these commands
to restart INETD to
activate new config:**

```
ENDTCPSVR SERVER(*INETD)
STRTCPSVR SERVER(*INETD)
```

31

SHOWTIME Program (1 of 3)

The program that gets submitted by INETD



```
D errMsg      s          52a
D cmd         s          20a  varying
D buf         s          20a
D data        s          26a
D len         s          10i  0
D mySock      s          10i  0

/free

mySock = 0;

exsr RecvCmd;

if cmd = 'SEND TIME';
  exsr SendTime;
endif;

callp close(0);

*INLR = *ON;
return;
```

INETD will always pass you the connected socket as descriptor #0. You do not have to call socket() to create it, it has been done for you by INETD.

32

SHOWTIME Program (2 of 3)

The program that gets submitted by INETD



```
begsr RecvCmd;

  cmd = '';
  dou ( %len(cmd)>1 and %subst(cmd:%len(cmd)-1:2) = x'0d0a' );

  len = recv( mySock: %addr(buf): %size(buf): 0);
  if (len = -1);
    errMsg = 'Error during recv.';
    leavesr;
  endif;

  cmd = cmd + %subst(buf: 1: len);
enddo;

// Translate to EBCDIC and strip off CRLF

buf = cmd;
QDCXLATE( %len(cmd) : buf: 'QTCPEBC' );
cmd = %subst(buf:1:%len(cmd));
%len(cmd) = %len(cmd) - 2;

endsr;
```

33

SHOWTIME Program (3 of 3)

The program that gets submitted by INETD



```
begsr SendTime;

  // send timestamp:

  data = %char(%timestamp(): *ISO);
  QDCXLATE( %len(data): data: 'QTCPASC' );

  len = send( mySock: %addr(data): %size(data): 0);
  if (len = -1);
    errMsg = 'Error during send.';
    callp close(mySock);
    leavesr;
  endif;

endsr;
```

```
CALL GETTIME PARM('server6.scottklement.com')
```

```
DSPLY Server's time is: 2007-04-03-12.19.47.692000
```

34

Application-Level Protocols



Once you have the basics of connecting, binding, accepting and spawning written once, you can create a template, and do it the same way every time.

The hard part is knowing what needs to be sent and received!

Just as different people speak different languages, so do computer programs. Each one has it's own set of commands and expected responses. This is called an "application-level protocol."

For the most part, the only thing that changes from one application to another is what gets sent and received!

RED TEXT = Sent by Server

BLACK TEXT = Sent by Client

35

Example Application: SMTP E-mail



```
220 mail.scottklement.com is ready!
HELO iseries.example.com
250 mail.scottklement.com Hello iseries.example.com [1.2.3.4],
pleased to meet you
MAIL FROM:<sklement@systeminetwork.com>
250 2.1.0 <sklement@systeminetwork.com>... Sender ok
RCPT TO:<example@scottklement.com>
250 2.1.5 <example@scottklement.com>... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
From: Scott Klement <sklement@systeminetwork.com>
To: Example Person <example@scottklement.com>
Subject: Hello there.

SOCKETS RULE!  YEAH!
.
250 2.0.0 1337fG6x032243 Message accepted for delivery
QUIT
221 2.0.0 grungy.dstorm.net closing connection
```

36

Example Application: HTTP



```
GET /index.html HTTP/1.1
Host: www.scottklement.com:80
Connection: close

HTTP/1.1 200 OK
Date: Tue, 03 Apr 2007 07:49:09 GMT
Server: Apache/2.0.52
Content-Length: 8294
Connection: close
Content-Type: text/html; charset=ISO-8859-1

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
  <meta name="Author" content="Scott Klement">
  <title>Scott Klement's web page</title>
  .
  .
</html>
```

37

Objective 3



WHERE TO FIND DOCUMENTATION

38

Where To Find Application Docs



The only difference between the ones I've shown you and other protocols is the exact commands and responses. They all pretty much work the same.

All of the Internet Standard Protocols are documented by Request For Comments (RFC) documents. Here are some of the more popular ones:

<i>Protocol</i>	<i>Description</i>	<i>RFC</i>
HTTP	Hypertext Transport Protocol (Web)	2616
FTP	File Transfer Protocol	959
SMTP	Simple Mail Transport Protocol	2821
POP3	Post Office Protocol	1939
Telnet	Basic Terminal Emulation	854-861
TN5250	5250 emulation over Telnet	1205

These are all publicly available. To get started, go to

<http://www.faqs.org/rfcs>

39

More Information



IBM Provides documentation in the Information Center – but it's oriented towards the ILE C programming language. In RPG, you must write your own:

- Prototypes
- Data structures
- Constants

Or download them from my site! <http://www.scottklement.com/presentations/>

The official IBM documentation for the socket API is found under:

Programming / APIs / APIs by Category / Unix-type / Sockets

Scott has a (V4, fixed-format oriented) tutorial about sockets on his Web site:

<http://www.scottklement.com/rpg/socketut/>

Scott has also written articles in his newsletter:

Introduction: article ID 51701
Error handling: article ID 51720
Server programming: article ID 51809
Server w/INETD: article ID 53182
Timing Out Sockets: article ID 53809

Tip: To read these articles, key the article ID into the search box, in the upper-right corner of www.SystemiNetwork.com

40

More Information



In System iNEWS magazine:

May 2006 issue, "TCP/IP and Sockets in RPG"

<http://www.systeminetwork.com/article/rpg-programming/tcpip-and-sockets-in-rpg-600>

Sept 2006 issue, "SSL Sockets from RPG? Of Course You Can!"

<http://www.systeminetwork.com/article/rpg-programming/ssl-sockets-from-rpg-of-course-you-can-709>

Scott's open source RPG software, written with sockets:

HTTPAPI (HTTP protocol)

<http://www.scottklement.com/httpapi/>

FTPAPI (FTP protocol)

<http://www.scottklement.com/ftpapi/>

TN5250 (written in C for Linux, not in RPG)

<http://tn5250.sourceforge.net>

41

This Presentation



You can download a PDF copy of this presentation from:

<http://www.scottklement.com/presentations/>

Thank you!

42