

Using the TSO/E REXX TCP/IP Socket API

Kenneth.Tomiak@Schunk-Associates.com

February 26, 2003

Session 2845

Trademarks and Copyrights

- Trademarks and Registered trademarks used in this presentation are the property of their respective owners.
- Copyright ©2003 Schunk and Associates, Inc.
- Permission is hereby granted to SHARE, Inc. to distribute this presentation to conference attendees and in the conference proceedings. All other rights reserved.
- The information presented here has not been subject to any formal review. Use with your own discretion. No warranty is expressed nor implied.

Visit www.Schunk-Associates.com to see this and other presentations made by the staff of Schunk and Associates, Inc. The latest and most up-to-date version is always found here.

Agenda

- JCL Primer
- REXX Primer
- TCP/IP Primer
- Process Flow
- Samples and model
- Socket() API
- REXXWHOO code snippets
- Questions and Comments

This session is intended to show you how to use the TSO/E REXX TCP/IP Socket() Application Programming Interface. Some familiarity with JCL, REXX, and TCP/IP are advantageous to coding complex client and server applications. A simple client application, Whois, will be used for demonstration purposes. This application will query a registrar's whois server and receive the information the server is willing to return.

A cursory overview of TSO/E, REXX, and TCP/IP will whet your appetite to review how you develop ad-hoc programs. It should also lay down just enough of a foundation that when you download the full open-source program code and try it on your system you understand what is happening.

In my opinion, REXX is the way to go to prototype logic flows, perform simple ad-hoc client applications, and just experiment with Socket programming. For more complex server and client applications you may need to code in a "compiled" language.

JCL Primer

- **IKJEFT01 versus IKJEFT1B versus IRXJCL**
- **SYSTSPRT**
- **SYSTSIN**
- **SYSPROC versus SYSEXEC**

There are three programs to choose from when running REXX in batch; IKJEFT01, IKJEFT1B, and IRXJCL. Each has its own good and bad points.

IKJEFT01 is the traditional program name used for the TMP. Look at any of your standard logon procedures and you are likely to see this name used. As long as you have the correct DD statements allocated the step will normally end with cond-code 00. The two required DD statements are SYSTSIN and SYSTSPRT. Failure to include these two DD statements will result in U102 abends. Processing of multiple commands included in the SYSTSIN stream ends after the last command in the input stream. REXX code can be found from either SYSPROC or SYSEXEC.

IKJEFT1B will normally end with the cond-code of your last command's return code. Processing of multiple commands included in the SYSTSIN stream ends with the first non-zero return code from any command in the input stream. REXX code can be found from either SYSPROC or SYSEXEC.

```

/**
/** DOC: THIS JOB CONNECTS TO A REGISTRAR WHOIS SERVER
/** AND QUERIES A DOMAIN'S REGISTRATION RECORD.
/**
/**STEP001 EXEC PGM=IKJEFT1B, PARM=' ',
/** REGION=8M, DYNAMNBR=75, TIME=1200
/**STEPLIB DD DISP=SHR, DSN=TCPIP. SEZALINK
/**SYSTSPRT DD SYSOUT=*
/**SYSEXEC DD DISP=SHR, DSN=MYUSERID. CLIST
/**SYSTSIN DD *
%REXXWHOO DOMAIN. TLD
/*
/**
/***TCPIP DD DISP=SHR, DSN=SYS1. PROCLIB(TCPIP)
/***SYSTCPD DD DISP=SHR, DSN=SYS1. TCPARMS(TCPDATA)
/***SYSEXEC DD DISP=SHR, DSN=MYUSERID. CLIST(REXXWHOO)
/**

```

If only one command is to be executed you may change the SYSTSIN DD statement from 'DD *' to 'DD DUMMY' and then place the command on the PARM operand.

IRXJCL will process only one command, passed via the PARM operand. Unlike the TMP programs, SYSTSIN is used as standard input for the command specified on the PARM operand. This program is limited to REXX only code, no TSO/E commands are allowed. REXX code must be found in SYSEXEC.

```

//*
//STEP001 EXEC PGM=IRXJCL, REGION=38M
//          PARM='BIGVAR'
//SYSTSPRT DD  SYSOUT=*
//SYSEXEC  DD  DISP=SHR, DSN=C547KTO.MSH.CLIST
//*        DD  DISP=SHR, DSN=C547KTO.MSH.CLIST(BIGVAR)
//SYSTSIN  DD  DUMMY
//*
```

The library containing SOCKET must be accessible, either as part of your system's LINKLIST or via a JOBLIB or STEPLIB statement.

You may need to provide the TCPDATA statements for the TCPIP stack on your system. You are limited to using one stack concurrently. In the above example I show the default location for the TCPIP stack's JCL so that you can search for your TCPDATA if required.

The percent symbol (%) instructs TSO/E to bypass the search for a load module named REXXWHOO and search the SYSPROC/SYSEXEC statements instead.

Add a jobcard if required on your system.

REXX Primer

- **/* REXX */**
- **Function calls**
- **Exception Handling**
- **Parse**
- **EBCDIC versus Ascii**

REXX Do's and Don'ts

Do start with a comment line containing the string 'REXX' if you will be using SYSPROC, otherwise TSO/E assumes the code to be a Command List (CLIST) and not REXX code. SYSEXEC is always REXX code.

Do not make your comment block pretty by stringing lots of asterisks together. '/****** REXX *****/' is a performance no-no. The REXX interpreter must look ahead on each subsequent asterisk to see if this is an end-comment pair. Instead, use another character for your level of comments, '/*----- REXX -----*/' is faster to interpret.

Functions can be Internal or External. Be aware that Internal functions might re-use variable names already used by your code or local versions of variable names. Well-behaved functions will use their own local versions of variables and return results to the calling statement.

Set up exception handling routines for testing and execution time debugging. Show the failing statement line number and the line(s) of code. An error is not always an error: deleting a file that may or may not exist before attempting to allocate the file as new may be something you want to ignore. Turn error handling off and on as you deem appropriate.

Parse is very powerful and more efficient than multiple SUBSTR/LEFT/RIGHT function calls. See '*REXX PARSE Templates: Exposed!*' to learn much more about Parsing.

EBCDIC to Ascii translation is a socket option. Plan ahead. CRLF = x'0D25' is not x'0D0A'. No native CODEPAGE support.

Variables may contain a finite number of characters. Note that REXX imposes no restriction on the maximum length of results. However, there is usually some practical limitation dependent upon the amount of storage available to the language processor. In a simple test I was able to create six variables holding more than five million characters, each, with a REGION=38M and using PGM=IRXJCL.

TCP/IP Primer

- **Four layers**
- **Socket concepts**
- **Stream versus Datagram versus Raw**

Application/Stack/NIC => -----<= NIC/Stack/Application

Application	Stack		NIC		NIC	Stack		Application
Process	Transport	Network	Datalink		Datalink	Network	Transport	Process
Program	TCP/UDP	IP/ICMP	Hardware		Hardware	IP/ICMP	TCP/UDP	Program
Initialize								Initialize
Socket								Bind/Listen
Connect/Accept								
Send/Write								Recv
Recv								Send/Write
Close								
Shutdown								Shutdown
Terminate								Terminate

Transport->Network types must agree, TCP must connect with TCP, UDP with UDP and ICMP with ICMP.

IPADDR:PORT time-wait default is two minutes. Must be unique. Keep connection open until all requests for that IPADDR:PORT are satisfied, then move on to the next one.

Stream (TCP) commands must complete and yield a status of success or exception.

Datagram (UDP) packets lack delivery status and may arrive in any order.
RAW (ICMP) requires APF authorization and thus are not supported in REXX.

A socket is also known as a file handle or pipe between processes.

Using a telephone call as an analogy:

You both subscribe with a local carrier (Process), the person you want to talk to has a phone number (bind), you pick up the phone (Socket), dial a number (connect), the other party hears the phone ring (listen), picks up their phone (accept), you carry on a conversation; while one of you talks (send/write) the other one listens (recv), until you say goodbye (close), and hang-up the phones (shutdown). And you can keep using your phone in this manner until you unsubscribe (terminate) service with your local carrier.

Process Flow

- **Process**
 - Initialize
 - Exchange data
 - Close
- **Terminate**

Server Flow

```
Set up with stack
Bind to an IPADDR:PORT
Listen
    Accept socket
        Receive data, act on request, send data
    Close
Loop back to listen
```

Client Flow

```
Set up with stack
Set up Socket
    Connect to host
        Send data, wait for response, receive data
    Close
Shutdown
Terminate
```

Once a socket is set up you can loop through connecting to a host, transferring data, and closing the connection. Once connected to a host you can loop through several data transfer requests.

Samples and Model

- **SEZAINST [IBM Communication Server]**
 - EZARXRSS
 - EZARXRSC
- **URLCHECK.rex [IBM REXX Development in Boeblingen, Germany]**
- **MAIL.pm [David Wood]**
- **Whois.pm [Chip Salzenberg, Dana Hudes]**
- **REXXMAIL.rex [Kenneth Tomiak]**

Samples

The server and client sample code that come with TCP/IP are an excellent starting point.

URLCHECK came with IBM Object REXX for the PC. This code was presented at SHARE in 1999. It ports easily to TSO/E. Once you get this working you have half of what a browser does. I used this code to help understand how to fetch data from a web server.

Then I moved on to using some Perl code to write my own Mail User Agent (MUA) to send reports via email. MAIL showed me how to send data to an email server. REXXMAIL became my cross between URLCHECK and MAIL. This code is not available because of the potential for abuse. I recommend you explore Lionel Dyck's XMITIP dialog if you really want to send email.

For explaining the TSO/E REXX TCP/IP Socket API I wanted to use an application that might have some use. REXXWHOO represents a port of a common Perl module, Whois, that shows the information a Registrar's whois server is willing to return. I recommend you look up only your own domain and then not too frequently.

With the increase in the number of registrars you may have to try several before you get the data you want.

EZASOKET CALL

```

getIBMopt = Left('GETIBMOPT',16)
one = Copies('00'x,3) || '01'x
buffer100 = Copies('00'x,100)
errno1 = Copies('00'x,4)
retcode1 = Copies('00'x,4)
Address ATTCHPGM 'EZASOKET getIBMopt one buffer100',
  'errno1 retcode1'
attchrc = c2x(retcode1)
if attchrc >< 0 then,
  Do
    Say "Did attach work or not, rc="attchrc
  End

```

This is the old way. Notice how you have to initialize variables to expected sizes. This particular call will return the names of all of the TCPIP stacks active on your system.

```

say 'TCP/IP image(s) detected:',
  C2d(SubStr(buffer100,1,4))
say ' Name   Version Status'
say ' _____'
buffer100_index = 0
Do I = 1 to C2d(SubStr(buffer100,1,4))
  image_entry = SubStr(buffer100,(buffer100_index * I + 5),12)
  Parse VAR image_entry 1 i_status +2 i_version +2 i_name
  say ' Left(i_name,8),
      Right(C2x(i_version),7),
      Right(C2x(i_status),6)
End

say "
say "Using the DEFAULT stack"
gethostname = Left('GETHOSTNAME',16)
name255 = Copies(' ',255)
namelen = Right(Copies('00'x,4) || d2c(Length(name255)),4)
errno2 = Copies('00'x,4)
retcode2 = Copies('00'x,4)
Address ATTCHPGM 'EZASOKET gethostname namelen name255',
  'errno2 retcode2'
attchrc = c2x(retcode2)

parse var name255 hostname '00'x other
say 'HostName:' hostname.'
Exit 0

```

Socket() API

- **Returned_data = Socket('verb', other_parameters)**
 - Process
 - Initialize, change, and close
 - Exchange data
 - Resolve names
 - Manage configuration, options and modes

Socket syntax

Socket('verb', optional-operands)

Most of the verbs are supported. No need to initialize variables to predetermined sizes.

Category	Verbs
Coding Calls to Process Socket Sets	Initialize, Socketsetlist, Socketset, Socketsetstatus, Terminate
Coding Calls to Initialize, Change, and Close Sockets	Accept, Bind, Close, Connect, Givesocket, Listen, Shutdown, Socket, Takesocket
Coding Calls to Manage Configuration, Options, and Modes	Fcntl, Getsockopt, Ioctl, Select, Setsockopt, Version
Coding Calls to Exchange Data for Sockets	Read, Recv, Recvfrom, Send, Sendto, Write
Coding Calls to Resolve Names for REXX Sockets	Getclientid, Getdomainname, Gethostbyaddr, Gethostbyname, Gethostid, Gethostname, Getpeername, Getprotobyname, Getprotobynumber, Getservbyname, Getservbyport, Getsockname, Resolve

Process

Initiate_TCPIP_Services:

```

/*-----*/
/* Initiate the Socket services */
/*-----*/
Returned_data = Socket('Initialize', 'RexxWhoo', 2)
parse var Returned_data s_rc s_subtask s_maxdesc s_servicename
socket_rc = s_rc
If s_rc = 0 then,
  Do
    Say s_servicename":"s_subtask "Initiated" s_maxdesc "sockets"
  End
Else,
  Do
    say Returned_data
    call error 'E', 40, 'Unable to Initiate SOCKET RexxWhoo' s_rc
  End
Return

```

Provide a name for your socket and specify how many socket entries you would like. The name will appear in the output of a 'netstat' command. Ask for a reasonable quantity of socket entries. In this program we will never query more than one domain per execution so two is plenty.

You will get back the number of socket entries defined and the name of the stack. This will be aligned with the TCPDATA found from the SYSTCPD statement or default search order used by TCP/IP.

```
TCPIP: RexxWhoo Initiated 2 sockets
```

GetHostId

```

Who_Am_I:
/*-----*/
/* Find IP address of client's machine */
/*-----*/
Returned_data = Socket('Gethostid')
Parse var Returned_data s_rc s_results
If s_rc <> 0 Then,
  Do
    Say Returned_data
    Call error 'E', 100, 'GetHostID failed' s_rc
  End
Else,

```

Parse GetHostId results

```
Do
  max_ip_addresses = 0
  Do while s_results <> ""
    max_ip_addresses = max_ip_addresses + 1
    Parse var s_results ip_address s_results
    client_ipaddr.max_ip_addresses = ip_address
    Say "MY HOSTID(s) =" max_ip_addresses '-' ip_address
  End
  client_ipaddr.0 = max_ip_addresses
End
Return
```

A server could use a hard-coded IPADDR to listen on or it could discover the IPADDR of the stack it is part of. A client would discover its IPADDR to aid in debugging or to pass it as part of the data stream.

Other services you might use before proceeding would aid in identifying who you are, what string or numeric values relate to services and protocols, and what name or IPADDR relate to some other host.

A machine can have more than one IPADDR assigned so you would Parse the returned data and take the appropriate action for each IPADDR. There could be a different hostname assigned to each IPADDR. You might get the list of Host Ids and then perform a GETHOSTBYID on each IPADDR.

```
MY HOSTID(s) = 1 - 146. 203. 245. 2
146. 203. 245. 2 resolved to mainframe. internal. mydomain. tld
In domain=mydomain. tld
```

Establish a Socket

```

Establish_A_Socket:
/*-----*/
/* Establish a socket */
/*-----*/
Returned_data = Socket('Socket', 'AF_INET', 'STREAM', 'TCP')
Parse var Returned_data s_rc s_results
If s_rc <> 0 then,
  Do
    Call error 'E', 60, 'SOCKET(SOCKET) rc='s_rc
  End
Else,
  Do
    say "Socket(STREAM)="Returned_data
    s_d = s_results
  End
Return

```

You then make the connection between the Socket entry and your program. Think of this as the DCB in your program matching up with a DD statement in your JCL. When you read and write to the socket the stack will know which ‘pipe’ to use based on the socket_descriptor (s_d). The socket_descriptor is an index to the Socket entry, similar to the DDNAME you would use in your program.

This presentation will cover ‘STREAM’ sockets with ‘TCP’. You might use ‘DATAGRAM’ and ‘UDP’ for less critical applications.

```

Socket (STREAM)=0 1
socket descriptor=1

```

Show Socket version

```

Show_Socket_Version:
/*-----*/
/* Show the version of the sockets */
/*-----*/
Returned_data = Socket('Version')
parse var Returned_data s_rc s_results
If s_rc = 0 then,
  Do
    Say "Socket version="s_results
  End
Else,
  call error 'W', 50, 'Unable to get SOCKET version' s_rc
Return

```

Another service will return the version of the Socket API. This is useful if are writing code for multiple releases of operating systems. Interrogate the Socket version before exploiting new features.

Socket version=REXX/SOCKETS CS V2R6 APR 17, 1998

Show mode

```

Show_Blocking_Mode:
/*-----*/
/* Determine blocking mode */
/*-----*/
Returned_data = Socket('Fcntl', s_d, 'F_GETFL')
parse var Returned_data s_rc s_results
If s_rc <> 0 then,
  Do
    call error 'W', 140, 'SOCKET(Fcntl) rc='s_rc
  End
Else,
  Do
    say "Running in "s_results "mode"
  End
Return

```

You can learn what mode the socket is operating in by using 'FCNTL'. Advanced Socket programmers may wish to switch modes.

Runni ng i n Blocki ng mode

Set Socket options

```

Set_Socket_Options:
/*-----*/
/* Turn on EBCDIC-Ascii conversion */
/*-----*/
Returned_data = Socket('SetSockOpt',s_d,'SOL_SOCKET','SO_ASCII','On')
Parse var Returned_data s_rc s_results
If s_rc = 0 then,
  Do
    Say "Socket will do EBCDIC to ASCII conversion"
  End
Else,
  call error 'E', 70, 'Unable to set SOCKET So_ASCii' s_rc
Return
Ascii x'OD0A' is EBCDIC x'OD25' !!!

```

EBCDIC to Ascii translation is required if you are writing multi-platform code.

Socket will do EBCDIC to ASCII conversion

Remember that CODEPAGE support is not available and x'0A' gets translated to something other than what is required. The carriage return – line feed value of x'0D0A' should be coded as x'0D25'. This will translate correctly and the Ascii platforms will be happy.

Connect

```

Connect:
Parse Arg whois_server
Returned_data = Socket('Connect', s_d, 'AF_INET' port whois_server)
parse var Returned_data s_rc s_results
If s_rc <> 0 then,
  Do
    say 'connect to server' whois_server "failed," Returned_data
    call error 'E', 120, 'SOCKET(CONNECT) rc='s_rc
  End
Else,
  Do
    say "Socket('Connect','s_d', 'AF_INET'",
      port whois_server) ="Returned_data
  End
Return 0

```

Open the pipe to a host.

```
Socket('Connect', 1, 'AF_INET' 43 whois.networksolutions.com) =0
```

A return code of zero means you are ready to transfer data.

Oh who, oh who, could the Official Registrar be? VeriSign (Network Solutions) should be used for the common TLDs; .com, .net, and .org. You may even find .edu entries are correct, although a search of the web shows educause.edu is the official Registrar. Country Code TLD, ccTLD, entries are handled by other Registrars. I found agreements on the InterNic website that show the name of the whois server they agreed to provide. If you want to lookup any domain you will need to build, and maintain, a table of Registrar's whois servers.

The sample whois.pm code shows .arpa domains searched from whois.arin.net and whois.nic.mil used for .mil domains. And more recently, a new set of TLD qualifiers are being used for special interest groups; .aero, .name, .biz, .info, .int, and .museum. All with their own Registrars.

Send data

```

SendCommand:
Parse Arg Socket, Command
Call check_ops
Returned_data = Socket('Write', s_d, Command '0D25'x)
Parse var Returned_data s_rc s_results
If s_rc <> 0 then,
  Do
    call error 'E', 230, 'SOCKET(WRITE) rc='s_rc
  End
Else,
  Do
    s_data_len = s_results
    If s_data_len < 1 then say "Writing nothing?" s_data_len
  End
Return Returned_data

```

Ascii x'0D0A' is EBCDIC x'0D25' !!!

Send will return the number of bytes written. If you are sending a large amount of data it is up to your code to loop through and send all of the data. This simple client never sends a large amount of data. See the IBM Server sample for the code to send a large amount of data or look at how the READ LOOP logic works and code something similar.

```
cmd=DONTCRYFORMEARGENTINA.COM =cmd
```

In the sample output presented above there is non-displayable data between .COM and =cmd. The two positions contain the EBCDIC carriage return-line feed characters x'0D25', ASCII x'0D0A'.

Peek at Socket

```

Peek_at_socket:
Returned_data = Socket('Recv', s_d, 1024, 'PEEK')
parse var Returned_data s_rc s_type s_port s_ip s_results
parse var Returned_data s_rc s_data_len s_data_text
If s_rc <> 0 then,
  Do
    call error 'W', 250, 'SOCKET(Peek) rc='s_rc
  End
Return Returned_data

```

Use Peek to determine how much data is waiting to be received.

Peeking at the buffer will retrieve the specified amount of data without removing it from the socket.

I use peek to let me know when there is no more data to receive. Compare your results with the expected results. You may need to wait longer between receives on a slow network.

Read loop

```

Read_from_socket:
Response = "
Inbuffer = 0
Do Forever
  Returned_data = peek_at_socket()
  If s_rc <> 0 then,
    Do
      say 'I tried to peek but had a problem:'s_rc
      say '::Returned_data::'
    Leave
  End
  If s_data_len = 0 then,
    Do
      say "I peeked and found nothing more to read."
    Leave
  End

```

RecvFrom

```

BytesRcvd = Socket('Recvfrom', s_d, s_data_len)
Parse var bytesrcvd s_rc s_type s_port s_ipaddr s_real_len rcvdata
If s_rc < 0 Then
  Do
    say s_rc '<= 0 so I am leaving read_from_socket' rcvdata
  Leave
  End
If s_rc <> 0 then,
  Do
    call error 'I', 242, 'SOCKET(RecvFrom) rc='s_rc
  End
  Say "Add another" s_real_len "bytes of data to the inbuffer."
  Inbuffer = inbuffer + s_real_len
  Response = Response || rcvdata
End

```

Packets will be returned in varying sizes based on server and network congestion.

```

Add another 1024 bytes of data to the inbuffer.
Add another 230 bytes of data to the inbuffer.
Add another 1021 bytes of data to the inbuffer.
I peeked and found nothing more to read.
Inbuffer=2275 and length of response=2275

```

Loop until you have all of the data you should receive.

The reverse of this could be used to send a large amount of data. Keep stripping off the amount of data your send correctly sent and try sending the remainder until everything has been sent.

Close

```
Close_socket:
Returned_data = Socket('Close', s_d)
parse var Returned_data s_rc s_results
If s_rc <> 0 then,
  Do
    call error 'W', 234, 'Close rc='s_rc
  End
Return
```

Close the connection and Shutdown the socket when you are done.

Shutdown

```
Shut_the_socket:
Parse Arg Socket
Returned_data = Socket('Shutdown', s_d, 'BOTH')
parse var Returned_data s_rc s_results
If s_rc <> 0 then,
  Do
    call error 'W', 232, 'Shutdown rc='s_rc
  End
Return
```

Shutdown closes the pipe between the client and the server.

Terminate

```
Terminate_TCPIP_Services:
Returned_data = Socket('Terminate')
Return
```

Terminate will disconnect the program from the stack.

Here is what was returned for DONTCRYFORMEARGENTINA.COM:

The Data in the VeriSign Registrar WHOIS database is provided by VeriSign for information purposes only, and to assist persons in obtaining information about or related to a domain name registration record. VeriSign does not guarantee its accuracy. Additionally, the data may not reflect updates to billing contact information. By submitting a WHOIS query, you agree to use this Data only for lawful purposes and that under no circumstances will you use this Data to: (1) allow, enable, or otherwise support the transmission of mass unsolicited, commercial advertising or solicitations via e-mail, telephone, or facsimile; or (2) enable high volume, automated, electronic processes that apply to VeriSign (or its computer systems). The compilation, repackaging, dissemination or other use of this Data is expressly prohibited without the prior written consent of VeriSign.

VeriSign reserves the right to terminate your access to the VeriSign Registrar WHOIS database in its sole discretion, including without limitation, for excessive querying of the WHOIS database or for failure to otherwise abide by this policy. VeriSign reserves the right to modify these terms at any time. By submitting this query, you agree to abide by this policy.

Registrant: Casa dos Domínios (DONTCRYFORMEARGENTINA-COM-DOM)
 Portal do Morumbi Sao Paulo, Sao Paulo 05641-900 Brazil 323-1353
 cddom142@hotmail.com Domain Name: DONTCRYFORMEARGENTINA.COM
 Administrative Contact: Casa dos Domínios cddom142@hotmail.com
 Portal do Morumbi Sao Paulo, Sao Paulo 05641-900 Brazil
 323-1353 Technical Contact, Zone Contact: Casa dos Domínios
 cddom142@hotmail.com Portal do Morumbi Sao Paulo, Sao Paulo
 05641-900 Brazil 323-1353 Record last updated on 25-
 Mar-2003. Record expires on 18-Apr-2003. Record created on 18-
 Apr-2000. Domain servers in listed order: ns1.dr-
 parkingservices.com 65.82.1.86 ns2.dr-parkingservices.com 65.82.1.87
 The previous information has been obtained either directly from the registrant or a registrar of the domain name other than Network Solutions. Network Solutions, therefore, does not guarantee its accuracy or completeness.

VeriSign uses a x'25' between blocks of information which are not visible in this display of output.

There can be different text messages returned from the same server for different domains.

Register or transfer domains at www.BuyDomains.com - as low as \$9/year. Including FREE: Responsive toll-free support, URL/frame/email forwarding, easy management.

The lack of a match does not mean the Domain is available. You might have used the wrong registrar whois server.

By submitting this query, you agree to abide by this policy. NO MATCH: This domain is available! Go to www.networksolutions.com to register it now!

Top Level Domains (TLD) and Country Code TLDs have different whois servers!

Summary

- **REXX is a quick prototype environment**
- **Sockets are easy to work with**
- **Great for ad-hoc clients**

Bibliography

- **OS/390 IBM Communications Server IP Application Programming Interface Guide**
- **OS/390 TSO/E REXX Reference**
- **OS/390 MVS JCL Reference**
- **<http://www.icann.org/tlds/agreements/pro/registry-agmt-appo-06may01.htm>**
- **<http://www.rfc.net/>**

Visit <http://www.schunk-associates.com/presentations> to download the full REXXWHOO code and Adobe PDF formats of this presentation and handout. They will appear under *Kenneth Tomiak – Share100*.

Possible uses:

Query a time server and show the difference between your system's time and the time server. Useful in trying to debug multi-platform applications whose platforms have their own time.

Write your own MVS based Web Browser. I suggest trying to use Document Composition Facility (Script) to format the html text. Change pfkeys to be cursor sensitive and act like a mouse!

URLCHECK can be used to watch for changed pages. Tweak it to store its output where your web server can retrieve it.

Connect to different IPADDR:PORTs within your intranet to validate your network is functioning.