DESCRIPTION

Winsock RCMD32.DLL is a Microsoft Win32 Dynamic Link Library that provides a Windows Sockets compatible function that allows you to execute commands on a remote host and retrieve the output of those commands for processing by your program.

Winsock RCMD32.DLL is similar to the ***rcmd*** library function found on many Unix systems. The DLL includes the main **WinsockRCmd** function which initiates the command, plus functions for receiving data over the connection from the remote host executing the command.

The Winsock RCMD32.DLL function calls are completely source code compatible with the 16-bit Winsock RCMD.DLL.

The remote host must be a system running the *rshd* or *rexecd* server process, such as a Unix system or a Windows system running Denicomp Systems' Winsock RSHD/95 or Winsock RSHD/NT.

Winsock RCMD32.DLL is designed to execute non-interactive remote commands. If you need to execute an interactive command on the remote host, use a utility like Telnet.

REQUIREMENTS

Winsock RCMD32.DLL requires a PC running a 32-bit Windows operating system and any programming language that supports calls to DLL functions, such as C, Visual Basic, or Powerbuilder.

INSTALLATION

To install Winsock RCMD32.DLL, create a directory where you would like it to be installed and simply copy the files from the diskette. From a command prompt, type:

```
COPY A:*.*
```

The directory where you install the file **RCMD32.DLL** must be included in your PATH environment variable or you can copy this file to your Windows directory (usually \WINNT or \WINDOWS) or Windows System directory (such as \WINNT\SYSTEM32).

This diskette contains the file **RCMD32.LIB**, which is the import library for RCMD32.DLL. C programmers must link this file to your programs to use RCMD32.DLL, unless you are going to load and unload RCMD32.DLL dynamically using LoadLibrary() and FreeLibrary().. You may move this file to the directory containing your other C libraries if you wish.

The diskette also contains the file **RCMD32.H**, which is a C header file containing the function declaration for Winsock RCMD32.DLL. C programmers will "#include" this file in programs. You may move this file to a directory your compiler searches for header files if you wish.

The diskette also contains an example C program: **CRSH**. The file CRSH.C is the source code, CRSH.MAK is the makefile, and CRSH.EXE is the compiled program.

The was written using Micrsoft Visual C/C++. To compile it with Visual C/C++, type:

```
nmake /f crsh.mak crsh.exe
```

The sample program is similar to the RSH command. Its syntax is:

```
crsh host user command
```

Where ***host*** is the host name of the remote system, ***user*** is the login name of a user on the remote host, and ***command*** is the command to be executed. The ***user*** parameter **must** be specified, unlike RSH. The ***command*** may contain spaces. However, if it contains any special characters that are interpreted by the Windows command shell, you must enclose the command in double quotes.

The output of the command, if any, will displayed on your screen. Remember, only non-interactive commands can be executed.

FUNCTION DESCRIPTIONS

FUNCTION: WinsockRCmd

C:

```
INT WinsockRCmd (Rhost, RPort, LocalUser, RemoteUser, Cmd, ErrorMsg, ErrorLen)
   LPSTR RHost;
   int RPort;
   LPSTR LocalUser;
   LPSTR RemoteUser;
   LPSTR Cmd;
   LPSTR ErrorMsg;
   int ErrorLen;
```

Visual Basic:

```
Declare Function WinsockRCmd Lib "RCMD32.DLL"
  (ByVal RHost As String,
   ByVal RPort As Long,
   ByVal LocalUser As String,
   ByVal RemoteUser As String,
   ByVal Cmd As String,
   ByVal ErrorMsg As String,
   ByVal ErrLen As Long) As Long
```

Powerbuilder:

```
Function Int WinsockRCmd
   (Ref String RHost, &
    Int RPort, &
    Ref String LocalUser, &
    Ref String RemoteUser, &
    Ref String Cmd, &
    Ref String ErrorMsg, &
    Int ErrLen)  Library "RCMD32.DLL"
```

Usage:

The **WinsockRCmd** function initiates a connection to the remote host and executes the specified command if access is permitted. You can then use the **RCmdRead** function to receive the standard output and standard error output of the command.

Parameters:

RHost:

Specifies the name of a remote host that is listed in the "hosts" file. You will receive an error if the host name is not found.

RPort:

Specifies the port to use for the connection. Normally, this is the well-known port number of 514, which is the *rshd* server on the remote host. You may also specify port number 512, which is the *rexecd* server. The difference is explained later.

You can specify the port number or use the Windows Sockets **getservbyname()** function if a *services* file is present.

LocalUser:

The user name of the user on the local host (the PC). This can be NULL or an empty string if you want Winsock RCMD32.DLL to determine the name. The method it uses is described later.

This name is sent as the local user to the *rshd* server on the remote host. In general, it should be the same as the ***RemoteUser*** parameter.

RemoteUser:

The user name to be used at the remote host. This must be a valid user name at the remote host. It can be NULL or an empty string if you want Winsock RCMD32.DLL to determine the name. The method is uses it described later.

If you are using the *rexecd* server instead of the *rshd* server, specify the user's password in this parameter. The *rexecd* server is explained later.

Cmd:

The command to be executed at the remote host.

ErrorMsg:

A pointer to an area that can be used to store an error message from **WinsockRCmd**. If an error occurs while connecting to the remote host, **WinsockRCmd** will return a negative result and will store an error message here. You can then optionally use the error message as diagnostic output in your program.

ErrorLen:

The maximum length of the error message returned. This is the number of characters available in ***ErrorMsg***.

Return Value:

If **WinsockRCmd** successfully connects to the remote host and begins executing the specified command, it will return an integer "handle" that must be used by the **RCmdRead**, **RCmdHandle**, and **RCmdClose** functions. This handle will always be greater than or equal to zero.

If **WinsockRCmd** is not successful, it will return a negative number. If the number is -1, an error internal to **WinsockRCmd** occurred and a message describing the error will be stored in ***ErrorMsg***. If the number is not -1, it is a Windows Sockets error code (but negative). ***ErrorMsg*** will also contain a message attempting to describe the error.

Notes:

The error message returned in ***ErrorMsg*** may contain newline characters (ASCII 10). The message is suitable for display using the standard Windows **MessageBox** function or the Visual Basic **MsgBox** command/function.

Execution of interactive commands (commands requiring keyboard input) is **not** supported.

Unlike the Unix **rcmd** function, **WinsockRCmd** does not have the ability to separate the standard output and the standard error output of the remote command. Output to the Unix standard output and standard error will be intermixed.

Note for Visual Basic and Powerbuilder Users:

If you are using Visual Basic, Powerbuilder, or any similar language that uses dynamically allocated string variables, you must **allocate space** in the string passed as the ***ErrorMsg*** parameter before calling **WinsockRCmd**. This is very important; your program will most likely abort with a Windows error when an error message needs to be returned.

In Visual Basic, you can do this with the **String$()** function. In Powerbuilder, you can do this with the **Space()** function.

For example, if you want to allow for 128 characters in the ***ErrorMsg*** parameter, before calling **WinsockRCmd** you would do the following:

```
errmsg$ = String$(128,Chr$(0))
```

In Powerbuilder, you would do this:

```
errmsg = Space(128)
```

Using the *rexecd* Server

If you wish to use the *rexecd* server instead of the *rshd* server, specify a port number of 512 instead of 514 in the **WinsockRCmd** function call.

The *rexecd* server is similar to the *rshd* server, except that it requires a password to be specified. You must make provisions in your software to allow the user to enter the password at the appropriate time or retrieve it from some storage area (i.e. an initialization file). Winsock RCMD32.DLL does **not** ask for the password.

When using *rexecd*, in addition to using port 512 instead of 514, you must pass the password in the ***RemoteUser*** parameter in the **WinsockRCmd** function call (the fourth parameter). All other parameters and the remaining functions described in this manual function in the same manner.

FUNCTION: RCmdRead

C:

```
INT RCmdRead (hRCmd, RData, RCount)
   int hRCmd;
   LPSTR RData;
   int RCount;
```

Visual Basic:

```
Declare Function RCmdRead Lib "RCMD32.DLL"
   (ByVal hRCmd As Long,
    ByVal RData As String,
    ByVal RCount As Long) As Long
```

Powerbuilder:

```
Function Int RCmdRead
   (Int hRCmd,  &
    Ref String RData, &
    Int RCount) Library "RCMD32.DLL"
```

Usage:

The **RCmdRead** function reads the output of the command executed with **WinsockRCmd**. This allows you to capture the standard output and standard error output of the command you executed.

Parameters:

hRCmd:

This is the integer "handle" returned from the **WinsockRCmd** function.

RData:

A pointer to an area of memory to hold the data received.

RCount:

The maximum number of bytes to receive. This should be between 1 and 8192. You can specify a number larger than 8192, but no more than 8192 bytes will be returned in any single call.

Return Value:

If **RCmdRead** is sucessful, it returns the number of bytes read. It will be a number greater than zero.

If **RCmdRead** returns zero, there is no more data to read. The command has ended and you should call **RCmdClose** to terminate the connection.

If **RCmdRead** returns a negative number, an error has occurred. The number will be the Windows Sockets error number (but negative). You should call **RCmdClose** even if an error occurs to free all resources used by the connection.

Important Note for Visual Basic Users:

You must reserve space in the Visual Basic string passed as the ***RData*** paramter that is to receive the data returned. You do this with the **String$** function. For example, if you wanted to read 64 bytes at a time into the variable **d$**, you would:

```
d$ = String$(64, Chr$(0))
result% = RCmdRead(hCmd%, d$, 64)
```

Afterwards, you should only use the number of bytes returned in result% in the string. Use the **Left$** function for this (i.e. Left$(d$,result%)).

Important Note for Powerbuilder Users:

Like Visual Basic, you must reserve space in the string passed as the ***RData*** paramter that is to receive the data returned. You do this with the **Space()** function. For example, if you wanted to read 64 bytes at a time into the variable **d**, you would:

```
d = Space(64)
result = RCmdRead(hCmd, d, 64)
```

Afterwards, you should only use the number of bytes returned in result in the string. Use the **Left()** function for this (i.e. Left(d,result)).

If you are using any other language that uses dynamically allocated strings, you must do something similar to these two examples.

FUNCTION: RCmdReadByte

C:

```
INT RCmdRead Byte(hRCmd)
  int hRCmd;
```

Visual Basic:

```
Declare Function RCmdReadByte Lib "RCMD32.DLL"
  (ByVal hRCmd As Long) As Long
```

Powerbuilder:

Function Int RCmdReadByte (Int hRCmd) Library "RCMD32.DLL"

Usage:

The **RCmdReadByte** function, like the **RCmdRead** function, reads the output of the command executed with **WinsockRCmd**. However, it instead reads one character at a time and returns each character as an integer return value.

The **RCmdReadByte** function is useful when it is not possible or not convenient to pass a pointer to an area of memory as required by **RCmdRead**.

Parameters:

hRCmd:

This is the integer "handle" returned from the **WinsockRCmd** function.

Return Value:

If **RCmdReadByte** is sucessful, it returns the next character read from the standard output or standard error of the command executed. It will be a positive number between 1 and 255.

If **RCmdReadByte** returns zero, there is no more data to read. The command has ended and you should call **RCmdClose** to terminate the connection.

If **RCmdReadByte** returns a negative number, an error has occurred. The number will be the Windows Sockets error number (but negative). You should call **RCmdClose** even if an error occurs to free all resources used by the connection.

USING ASYNCHRONOUS READS

The **RCmdRead** and **RCmdReadByte** functions normally *block* (don't return) until there is either some data received, the remote command ends, or there is some TCP/IP error that closes the connection. This means that your program will be unresponsive while these functions are waiting for data to arrive.

You can execute the RCMD32.DLL functions in a separate thread in your program (separate from the user interface thread) so your program remains responsive to the user.

You can also use a Windows Sockets (Winsock) function to switch to *asynchronous reads*. This will cause Winsock to send messages to your program when data arrives or the connection is closed (i.e. the remote command ends) instead of having your program loop on blocking **RCmdRead** calls.

To do this, you must call the Winsock function **WSAAsyncSelect** and tell it to send messages on read events and close events. This function is documented in the Windows Sockets documentation, but here is a brief description of its parameters and usage:

***C*: int WSAAsyncSelect (SOCKET s, HWND hWnd, unsigned int wMsg, long lEvent)**

***Visual Basic*: Declare Function WSAAsyncSelect Lib "wsock32.dll"**
ByVal s As Long, ByVal hWnd As Long, ByVal wMsg As Integer,
ByVal lEvent As Long) As Long

After calling the **WinsockRCmd** function to establish the connection, an example would be:

res = WSAAsyncSelect(RCmdHandle(hRCmd), m_hWnd, WM_USER+1, FD_READ + FD_CLOSE);

After this call, your program will receive the message **WM_USER + 1** whenever data is available for reading from the connection or the connection is closed (the command ends or an error occurs). You can substitute any message value for **WM_USER + 1**, but that is used in this example.

When you receive the **WM_USER + 1** message, check the lower word of the **lEvent** message parameter to determine if you are receiving a read event or a close event. For example:

```
case WM_USER + 1:
       switch (LOWORD(lEvent))
      {
            case FD_READ:
                do {
                     res = RCmdRead(hRCmd, lpData, 128);
                     process_data(lpData);
                } while (res > 0);
                break;
            case FD_CLOSE:
                res = WSAAsyncSelect(RCmdHandle(hRCmd), m_hWnd, 0,0);
                res = RCmdClose(hRCmd);
                break;
      }
```

When the FD_READ event is received, you must call **RCmdRead** until it returns a value less than or equal to zero, since you do not know how much data is available to be read. When you have read all data that is available, you will receive a return value of **-10035** from **RCmdRead**. This is the Winsock error WSAEWOULDBLOCK, meaning that the receive operation normally would have blocked because no data is available. You should **not** treat this error code as a true error condition. It simply means that at this time, there is no data available to be read and you should wait until the next FD_READ event occurs for more data.

Note the call to **WSAAsyncSelect** on the close event (FD_CLOSE). This stops Winsock from sending any other messages and the connection can then be closed. Remember, receiving the FD_CLOSE event means that you should close the connection with **RCmdClose**. Winsock does not do it for you.

The asynchronous method can easily be used in the C language, but to use that method in a language such as Visual Basic, you will need to use a third party control that allows you to receive and process messages, such as Message Blaster 32. Visual Basic itself only allows you to receive pre-defined messages.

FUNCTION: RCmdClose

C:

```
INT RCmdClose (hRCmd)
   int hRCmd;
```

Visual Basic:

```
Declare Function RCmdClose Lib "RCMD32.DLL"
   (ByVal hRCmd As Long) As Long
```

Visual Basic:

```
Function Int RCmdClose (Int hRCmd) Library "RCMD32.DLL"
```

Usage:

The **RCmdClose** function closes the connection to the remote host and frees all resources used by the connection. You should call **RCmdClose** for each successful use of the **WinsockRCmd** function.

Parameters:

hRCmd:

This is the integer "handle" returned from the **WinsockRCmd** function.

Return Value:

If **RCmdClose** is sucessful, it returns zero. If it is unsuccessful, it returns a negative number that is the Windows Sockets error number (but negative).

FUNCTION: RCmdHandle

C:

```
INT RCmdHandle (hRCmd)
    int hRCmd;
```

Visual Basic:

```
Declare Function RCmdHandle Lib "RCMD32.DLL"
  (ByVal hRCmd As Long) As Long
```

Powerbuilder:

Function Int RCmdHandle (Int hRCmd) Library "RCMD32.DLL"

Usage:

The **RCmdHandle** function returns the Windows Sockets handle ("socket") for the connection, which can be used to call Windows Socket functions that require a SOCKET parameter.

Parameters:

hRCmd:

This is the integer "handle" returned by the **WinsockRCmd** function.

Return Value:

RCmdHandle returns a Windows Sockets handle. If you call **RCmdHandle** on a **WinsockRCmd** connection that is not opened, the return value is undefined.

DETERMINING THE USER NAME

You can pass a NULL in either the ***LocalUser*** or ***RemoteUser*** parameters in the **WinsockRCmd** function call and Winsock RCMD32.DLL will determine the user name. The following **only** applies if you pass a NULL in one of these parameters.

The local user name is normally the name you used when logging in to Windows. For example, if you logged in to Windows as the user "joed", **WinsockRCmd** will use "joed" as the user name at the remote host. **WinsockRCmd** will always convert this name to all lowercase characters.

WinsockRCmd will also look at the file **WIN.INI** in the Windows directory (e.g. \WINNT or \WINDOWS) for an alternate user name.

If **WIN.INI** contains a section named "[RCMD]" and contains an entry named "User" in that section, the name specified there will be used as the local user name. For example, **WIN.INI** might contain:

[RCMD]
User=joe

If this appeared in **WIN.INI**, the local user name would be "joe" and **WinsockRCmd** would use this name at the remote host.

To support multiple users, **WinsockRCmd** will also look for a section named "[RCMD-user]" in **WIN.INI** first for an alternate user name, where the "user" in the section name is the name used to log in to Windows. WinsockRCmd will look at this section first; if it does not exist, it will then look at the "[RCMD]" section.

For example, if "maryk" and "joed" are both users on the Windows PC, but their user names at the remote host are "mary" and "joe" respectively, **WIN.INI** might look like this:

[RCMD-joed]
User=joe

[RCMD-maryk]
User=mary

When the Windows user "joed" runs a program using **WinsockRCmd**, "joe" will be used at the remote host. When the Windows user "maryk" runs the program, "mary" will be used instead.

SECURITY

The remote host allows access only if at least one of the following conditions is satisfied:

? The name of the local host is listed as an equivalent host in the */etc/hosts.equiv* file on the remote host. This name must also be listed in the */etc/hosts file* on the remote host along with the proper IP address.

? If the local host is not in the */etc/hosts.equiv* file, the user's home directory on the remote host must contain a *.rhosts* file that lists the local host and user name. The *.rhosts* file in the user's home directory must be owned by either the user specified or root, and only the owner should have read and write access. That is, it **must** have permissions of **0600**.

? The user's login on the remote host does not require a password.

? A valid password has been supplied if using the *rexecd* service (port 512).

These requirements are defined by the remote shell daemon (*rshd)* or remote execution daemon (*rexecd)* running on the remote host, not by Winsock RCMD32.DLL. For more details, you can review the documentation for those daemons on the host system. For example, if the host is a Unix system, you can type "man rshd" to view the manual pages for the remote shell daemon.

EXAMPLES

For a complete example of the use of **WinsockRCmd** in C, see the **CRSH** program provided in the distribution.

For an example of the use of **WinsockRCmd** in Visual Basic, see the **VRSH** program provided in the distribution.

```
// Assumes that the host is in "rhost", the user is in "ruser", and the
// command is in "cmd".

    int hRCmd;
    char c;
    int result;

    hRCmd = WinsockRCmd(rhost,514,ruser,ruser,cmd,errmsg,sizeof(errmsg));

    if (hRCmd < 0)
      MessageBox(NULL,errmsg,"Remote Shell",MB_OK);
    else
    {
     while((result = RCmdRead(hRCmd,&c,128)) > 0)
              DisplayChars(c, result);
    }
    RCmdClose(hRCmd);
```

SUPPORT

Support is available via E-Mail:

Internet: support@denicomp.com
WWW: http://www.denicomp.com