**Resolved [RESOLVED] Sending and Receiving Byte Data Using Winsock**

Client Code:

Code:

Dim b() As Byte

Dim DataSent As Boolean

Private Sub Form\_Load()

Open App.Path & "\mypicture.jpg" For Binary Access Read As #1

ReDim b(0 To LOF(1) - 1)

Get #1, , b

Close #1

End Sub

Private Sub Command1\_Click()

Winsock1.Close

Winsock1.Connect "xxx.xxx.xxx.xxx", 8888

End Sub

Private Sub Winsock1\_Connect()

DataSent = False

Winsock1.SendData b

Do While DataSent = False

DoEvents

Loop

Winsock1.Close

End Sub

Private Sub Winsock1\_SendComplete()

DataSent = True

End Sub

Server Code:

Code:

Option Explicit

Dim Buffer() As Byte

Private Sub Command1\_Click()

Winsock1.Close

Winsock1.LocalPort = 8888

Winsock1.Listen

End Sub

Private Sub Winsock1\_Close()

Open App.Path & "\BYTE\_ARRAY.BYT" For Binary As #1

Put #1, 1, Buffer

Close #1

End Sub

Private Sub Winsock1\_ConnectionRequest(ByVal requestID As Long)

Winsock1.Close

Winsock1.Accept requestID

Buffer = ""

End Sub

Private Sub Winsock1\_DataArrival(ByVal bytesTotal As Long)

Winsock1.GetData Buffer

End Sub

## send hex data to Ethernet or LAN using Visual basic 6.0

Private Sub Command1\_Click()

Dim boInvalid As Boolean

Dim intI As Integer

Dim intJ As Integer

Dim strHex As String

Dim bytToSend() As Byte

'

' Must be an even number of characters entered

'

If Len(txtdata.Text) Mod 2 = 0 Then

ReDim bytToSend((Len(txtdata.Text) \ 2) - 1)

intI = 1

Do

strHex = UCase(Mid$(txtdata.Text, intI, 2))

'

' Each character must be A thru F or 0 thru 9

'

For intJ = 1 To 2

Select Case Mid$(strHex, intJ, 1)

Case "A" To "F", "0" To "9"

Case Else

boInvalid = True

End Select

Next intJ

'

' If it's a valid pair of characters

' convert the hexadecimal representation to a binary value

' and store in the next element of the array

' If it's invalid, tell the user and stop processing

'

If Not boInvalid Then

bytToSend(intI \ 2) = Val("&H" & Mid$(txtdata.Text, intI, 2))

intI = intI + 2

Else

MsgBox "Invalid Hexadecimal Value: " & strHex

End If

Loop Until intI > Len(txtdata.Text) Or boInvalid

'

' If it was valid hexadeciomal data then send it

'

If Not boInvalid Then

Winsock1.SendData bytToSend

End If

Else

MsgBox "Invalid Data - Hexadecimal data must be an even number of characters"

End If

End Sub

Private Sub cmdSendHexString\_Click()

If Len(Text1.Text) Mod 2 <> 0 Then

MsgBox "Must be even number of hex codes"

Text1.SetFocus

Else

'

' Code to send hex string or convert to actual hex

'

End If

End Sub

Private Sub Text1\_KeyPress(KeyAscii As Integer)

Select Case True

Case Chr$(KeyAscii) Like "[!0123456789AaBbCcDdEeFf]"

KeyAscii = 0

End Select

Exit Sub

## [RESOLVED] Receive Certain data useing Winsock VB6

Private Sub tcpProgramServer\_DataArrival(ByVal bytesTotal As Long)

Dim strData As Byte

For i = 0 To bytesTotal

tcpProgramServer.GetData strData, vbByte

Text1.Text = Text1.Text & " " & Right$("0" & Hex$(strData), 2)

Next i

End Sub

## [RESOLVED] converting winsock recieved hex data to string

Code:

Private Sub winsock1\_DataArrival(ByVal sender As System.Object, ByVal e As AxMSWinsockLib.DMSWinsockControlEvents\_DataArrivalEvent) Handles winsock1.DataArrival

Dim dat As String 'where to put the data

' Dim data As Byte() = System.Text.UnicodeEncoding.ASCII.GetString(dat)

winsock1.GetData(dat, vbString, 1000) 'writes the new data in our string dat ( string format )

'add the new message to our chat buffer

Form2.TextBox1.Text = Form2.TextBox1.Text & vbNewLine &"Server : " & dat & vbCrLf

End Sub

Code:

Private Sub winsock1\_DataArrival(ByVal sender As System.Object, ByVal e As AxMSWinsockLib.DMSWinsockControlEvents\_DataArrivalEvent) Handles winsock1.DataArrival

Dim dat(e.bytesTotal - 1) As Byte '<<< declare a byte array to store the arriving bytes

winsock1.GetData(dat, vbByte, dat.Length) ' get the data

'add the new message to our chat buffer

Dim msg As String = System.Text.Encoding.UTF8.GetString(dat)

Form2.TextBox1.Text = Form2.TextBox1.Text & vbNewLine &"Server : " & msg & vbCrLf

End Sub

**Winsock Packet Buffer**

Hi there. Googling brings me here a lot, so I figured I'd share some code that is nearly impossible to find information on, as well as ask for some suggestions.  
  
This has to do with buffering TCP data in Winsock. There is a surprising lack of code going around about this subject, and most of it is about using packet delimiters. Well, sometimes you aren't creating the server and simply can't do this.  
  
In my example there are no packet delimiters, but I know the length each packet should be based on the first byte. What I'm looking for are suggestions to make the code better, as this seems like an inefficient way to do things.  
  
Declare a global variable to store incoming data

Code:

Private mainBuffer as string

Function to convert a string to hex (I believe I snagged this from these forums, actually)

Code:

Public Function StringToHex(ByVal StrToHex As String) As String

Dim strTemp As String

Dim strReturn As String

Dim I As Long

For I = 1 To Len(StrToHex)

strTemp = Hex$(Asc(Mid$(StrToHex, I, 1)))

If Len(strTemp) = 1 Then strTemp = "0" & strTemp

strReturn = strReturn & strTemp & Space$(1)

Next I

StringToHex = strReturn

End Function

As data comes in, add it to the end of our buffer string

Code:

Private Sub wsock\_DataArrival(ByVal bytesTotal As Long)

Dim data As String, hexData As String

wsock.GetData data

hexData = StringToHex(data)

mainBuffer = mainBuffer & hexData

ProcessData

End Sub

Now I look at the first byte in the buffer, determine the length that the packet should be, and if I've have all the data, pass only the packet data off to a function to process, and remove that piece from the buffer.  
You'll see some multiplication and division by 3, that's because each byte is a total of 3 characters (2 characters for the actual byte, and a space to separate them)

Code:

Private Sub ProcessData()

Dim packetLen As Integer, packet As String

packetLen = 0

Do Until Len(mainBuffer) / 3 < packetLen

Select Case Left(mainBuffer, 2)

Case "00"

packetLen = 131

Case "01"

packetLen = 1

Case "02"

packetLen = 1

Case "03"

packetLen = 1028

Case "04"

packetLen = 7

Case "05"

packetLen = 9

Case "06"

packetLen = 8

Case "07"

packetLen = 74

Case "08"

packetLen = 10

Case "09"

packetLen = 7

Case "0A"

packetLen = 5

Case "0B"

packetLen = 4

Case "0C"

packetLen = 2

Case "0D"

packetLen = 66

Case "0E"

packetLen = 65

End Select

If Len(mainBuffer) / 3 >= packetLen Then

packet = Left(mainBuffer, packetLen \* 3)

mainBuffer = Right(mainBuffer, Len(mainBuffer) - packetLen \* 3)

ProcessPacket (packet)

End If

Loop

End Sub

Now, this code actually works, but it feels like I'm doing something terribly inefficient, redundant, and/or just stupid. Anyone care to point out the flaws here?

**Re: Winsock Data Arrival issue**

in just downloading I parse it all on Winsock Close .. but not sure if that will work in a server project .. once it closes I then handle the response codes, and if its downloading anything other than Text i create a file using the binary.  
  
Also I get the length from the header if I can ..  
small example .. though it may not apply to what you're doing ..?  
I used this on a couple 180mb files last night ..

VB Code:

1. Private ReceiveData         As String
2. Private FileLength          As Long
3. Private FileBytes           As Long
4. Private LengthParsed        As Boolean
6. '// GET DATA
7. Public Sub WinsockReceive(ByVal bytesTotal As Long)
8. Dim strData As String
9. On Error GoTo Err:
10. '// GET DATA
11. fWinsock.Winsock1.GetData strData, vbString
12. ReceiveData = ReceiveData & strData
13. FileBytes = FileBytes + bytesTotal
14. '// GET THE HEADER
15. If Not LengthParsed Then
16. GetLength
17. Else
18. '// DOWNLOAD FILE
19. Download
20. End If
21. Exit Sub
22. Err:
23. '// ERROR HANDLING
24. UpdateStatus ("Data Error")
25. WinsockDisconnect
26. Exit Sub
27. End Sub
29. '// GET FILE LENGTH
30. Private Sub GetLength()
31. Dim BreakePosition As Integer
32. Dim vHeaders As Variant
33. Dim vHeader As Variant
34. BreakePosition = InStr(1, ReceiveData, vbCrLf & vbCrLf)
35. If BreakePosition Then
36. LengthParsed = True
37. FileBytes = FileBytes - BreakePosition - 3
38. vHeaders = Split(Left(ReceiveData, BreakePosition - 1), vbCrLf)
39. '// GET THE CONTENT LENGTH
40. For Each vHeader In vHeaders
41. If InStr(1, vHeader, "Content-Length") Then
42. FileLength = CLng(Mid(vHeader, InStr(1, vHeader, " ") + 1))
43. Exit For
44. End If
45. Next
46. End If
47. End Sub
49. '// DOWNLOAD
50. Private Sub Download()
51. If FileLength > 0 Then
52. UpdateProgress
53. UpdateStatus ("Downloading: " & FileBytes & " of " & FileLength & " bytes " & \_
54. "(" & CInt(FileBytes / (FileLength / 100)) & "%)")
55. Else
56. UpdateStatus ("Downloading: " & FileBytes & " bytes")
57. End If
58. End Sub
60. '// UPDATE PROGRESS
61. Private Sub UpdateProgress()
62. fWinsock.ProgressBar1.Value = FileBytes / (FileLength / 100)
63. End Sub
65. '// UPDATE STATUS
66. Private Sub UpdateStatus(ByVal StatusType As String)
67. fWinsock.Caption = StatusType
68. End Sub

something i use found at vbip.com ..

VB Code:

1. '.... in winsock close ..
3. ReceiveHeader = Left$(ReceiveData, InStr(1, ReceiveData, vbCrLf & vbCrLf) + 1)
4. ReceiveData = Mid(ReceiveData, InStr(1, ReceiveData, vbCrLf & vbCrLf) + 4)
6. If GetHttpHeaderFieldValue(ReceiveHeader, "Transfer-Encoding") = "chunked" Then
7. ReceiveData = DecodeChunkedMessage(ReceiveData)
8. End If


12. Private Function GetHttpHeaderFieldValue(strHttpHeader As String, strHttpHeaderField As String) As String
13. Dim strBuffer As String
14. Dim intStart As Integer
15. Dim strSearchString As String
16. strSearchString = vbCrLf & strHttpHeaderField & ": "
17. intStart = InStr(1, strHttpHeader, strSearchString) + Len(strSearchString)
18. strBuffer = Mid$(strHttpHeader, intStart, InStr(intStart, strHttpHeader, vbCrLf) - intStart)
19. If Len(strBuffer) > 0 Then
20. GetHttpHeaderFieldValue = strBuffer
21. End If
22. End Function
24. Private Function DecodeChunkedMessage(strMessage As String) As String
25. 'This is a scheme of chunked message
26. '<CHUNK SIZE><CRLF><DATA CHUNK><CRLF><CHUNK SIZE><CRLF><DATA CHUNK>...<0 CHUNK SIZE>
27. Dim lngPosA As Long
28. Dim lngPosB As Long
29. Dim intOctetsToRead As Integer
30. Dim strTempBuffer As String
31. Const CRLF\_LENGHT = 2
32. lngPosA = InStr(1, strMessage, vbCrLf)
33. intOctetsToRead = Val("&H" & Left(strMessage, lngPosA - 1))
34. Do Until intOctetsToRead = 0
35. strTempBuffer = strTempBuffer & Mid(strMessage, lngPosA + CRLF\_LENGHT, intOctetsToRead)
36. lngPosB = lngPosA + CRLF\_LENGHT + intOctetsToRead + CRLF\_LENGHT
37. lngPosA = InStr(lngPosB, strMessage, vbCrLf)
38. intOctetsToRead = Val("&H" & Mid(strMessage, lngPosB, lngPosA - lngPosB))
39. Loop
40. DecodeChunkedMessage = strTempBuffer
41. End Function

What you need to do is split down your chunks and then work out if the final chunk is the correct length (from the chunk header). If it isnt then store it in a static variable until the next packet arrives. Add the remainder of the last message to the front of the new one and bingo its in order.

VB Code:

1. Private Sub Winsock1\_dataarrival(BytesTotal as integer)
2. Static Remainder as string
3. Dim strData as string
5. 'Get your new packet
6. winsock1.getdata strData
8. 'Merge the remainder of the last one to this new one
9. strData = Remainder & strdata
11. 'Break down you message and until no more chunks are complete
13. 'Move any data from an uncompleted chunk to the remainder variable for the next arrival event
14. Remainder = (RestOfChunk) 'Whatever was left unprocessed

17. end Sub

I know this is an old post I'm replying to, but I've been using the above DecodeChunkedMessage code in a project of mine for a while, and for the first time yet it's encountered a block size over 8192. I've managed to isolate the problem down to this one specific line:

vb Code:

1. intOctetsToRead = Val("&H" & Left(strMessage, lngPosA - 1))

Update: In case no-one responds to this and someone else needs this code too, I've written a quick hex to decimal converter:

vb Code:

1. Public Function hextodec(str As String) As Long
2. Dim vl As Long, n As Long, b As Double
3. n = 1
4. For b = Len(str) To 1 Step -1
5. vl = vl + (Val("&H" & Mid(str, b, 1)) \* n)
6. n = n \* 16
7. Next b
8. hextodec = vl
9. End Function

**hould vb6 Strings be converted to byte arrays when using Com port communication?**

Is it good practice to always convert a vb6 string to a byte array when using Com.output?  
  
Would a string that was converted to a byte array like this:

Code:

Dim Sometext As String

Dim b() As Byte

Sometext = "Hello World"

b = StrConv(Sometext ,vbFromUnicode)

Be a better approach than simply do this:

Code:

Dim Sometext As String

Sometext = "Hello World"

Com.output = Sometext

The problem i'm facing is that 98% of commands work. But the device i send commands to, also has an 'internal hardware' that commands can be sent to, to do specific operations. These 'Specifics' dont work, or fail 99.8% of the time (specific number, but kinda accurate)  
  
  
It **doesn't work on my machine**using vb6, but it does work on the **manufacturer's machine, using .net**  
  
If we 'sniff' the data (using a 3rd party serial port monitor), everything seems to be exactly as we are sending it. ( *Although it doesn't work here, and it DOES work over there*.)  
  
So to exclude the hardware being the problem, we've sent the whole package to the manufacturer. And using our hardware it works there, while it didn't over here.

There is no endianess at question here. or is there? it's confusing..  
  
Literally the same command that is sent over the output is working for an .net application but isn't in my vb6 application.  
  
This string for example (in hex for readability): "530008000460EBAA"  
  
is sent to the output as follows:

Code:

str$ = chr(53) & chr(00)& chr(08)& chr(00)& chr(04)& chr(60)& chr(EB)& chr(AA)

com.output = str$

Can the result on the other end be different when it was sent like this, in other words, will it be more true to the actual data:

Code:

dim b() as byte

str$ = chr(&h53) & chr(&h00) & chr(&h08) & chr(&h00) & chr(&h04) & chr(&h60) & chr(&hEB) & chr(&hAA)

b = StrConv(str$ ,vbFromUnicode)

com.output = b

Instead one should use ones own converter-routine for such Hex-Stream-Inputs, like the one below:

Code:

Private Sub Form\_Load()

DumpBytesHex HexStrToByteArr("530008000460EBAA")

DumpBytesHex HexStrToByteArr("53 00 08 00 04 60 EB AA")

End Sub

Public Function HexStrToByteArr(strHex As String) As Byte()

Dim B() As Byte, i As Long, j As Long, B15 As Byte

B = Replace(strHex, " ", "") 'remove potential SpaceChars beforehand

For i = 0 To UBound(B) Step 2 'B contains a WChar-Stream, so we step with 2

Select Case B(i) + 256& \* B(i + 1) 'and re-construct the WChar-Value as Select-Case-Input

Case 48 To 57: B15 = B(i) - 48

Case 65 To 70: B15 = B(i) - 55

Case 97 To 102: B15 = B(i) - 87

Case Else: Err.Raise vbObjectError, , "invalid char in hex-stream"

End Select

If i Mod 4 Then B(j) = B(j) \* 16 + B15: j = j + 1 Else B(j) = B15

Next

If j Then ReDim Preserve B(j - 1)

HexStrToByteArr = B

End Function

Public Sub DumpBytesHex(B() As Byte) 'to visualize content of short ByteArrays

Dim i As Long

For i = 0 To UBound(B)

Debug.Print Right("0" & Hex(B(i)), 2); IIf(i = UBound(B), vbLf, " ");

Next

End Sub

Here is a simple trick that will get rid of most locale issues. Just \*always\* pass 3-rd parameter to StrConv and \*always\* read/write byte-arrays from/to the serial port.

Code:

Const LOCALE\_ID As Long = 1033

Dim b() As Byte

b = StrConv("this is a test - това е проба", vbFromUnicode, LOCALE\_ID)

Another (better) option would be to use conversion from/to UTF-8 not just en-US locale 1033 with some helper functions like these:

Code:

'--- for MultiByteToWideChar

Private Const CP\_UTF8 As Long = 65001

Private Declare Function MultiByteToWideChar Lib "kernel32" (ByVal CodePage As Long, ByVal dwFlags As Long, lpMultiByteStr As Any, ByVal cbMultiByte As Long, lpWideCharStr As Any, ByVal cchWideChar As Long) As Long

Private Declare Function WideCharToMultiByte Lib "kernel32" (ByVal CodePage As Long, ByVal dwFlags As Long, ByVal lpWideCharStr As Long, ByVal cchWideChar As Long, ByVal lpMultiByteStr As Long, ByVal cchMultiByte As Long, ByVal lpDefaultChar As Long, ByVal lpUsedDefaultChar As Long) As Long

Public Function ToUtf8Array(sText As String) As Byte()

Dim baRetVal() As Byte

Dim lSize As Long

lSize = WideCharToMultiByte(CP\_UTF8, 0, StrPtr(sText), Len(sText), 0, 0, 0, 0)

If lSize > 0 Then

ReDim baRetVal(0 To lSize - 1) As Byte

Call WideCharToMultiByte(CP\_UTF8, 0, StrPtr(sText), Len(sText), VarPtr(baRetVal(0)), lSize, 0, 0)

Else

baRetVal = vbNullString

End If

ToUtf8Array = baRetVal

End Function

Public Function FromUtf8Array(baText() As Byte) As String

Dim lSize As Long

If UBound(baText) >= 0 Then

FromUtf8Array = String$(2 \* UBound(baText), 0)

lSize = MultiByteToWideChar(CP\_UTF8, 0, baText(0), UBound(baText) + 1, ByVal StrPtr(FromUtf8Array), Len(FromUtf8Array))

FromUtf8Array = Left$(FromUtf8Array, lSize)

End If

End Function

The OP is now trying to convert this Hex-Input "80" to a ByteArray (with a single Byte and the value 128).  
And he does this via:

Code:

Dim S As String, B() As Byte

S = Chr(&H80) 'implicit ANSI to Unicode-conversion

B = StrConv(S, vbFromUnicode) 'explicit Unicode to ANSI conversion

Debug.Print B(0) '<- Prints out decimal 128 == &H80

This ANSI-Unicode + Unicode-ANSI backconversion works (although still risky),  
because both parts use the same (default-)locale for the ANSI-mapping-table.  
  
But if we introduce ChrW as the first conversion-function:

Code:

Dim S As String, B() As Byte

S = ChrW(&H80) 'UTF16-UnicodeValue to Unicode-WChar-conversion

B = StrConv(S, vbFromUnicode) 'explicit Unicode to ANSI conversion

Debug.Print B(0) '<- Prints out decimal 63 <> &H80

Of course the task would be quite easy, when the OP would just do:  
Redim B(0) As Byte  
B(0) = &H80  
  
Or when the whole thing is part of a (Step 2)-loop over the HexLiteral-InputString, then:  
Redim B(0 to Len(HexInput) \ 2 - 1) As Byte  
B(i \ 2) = CByte("&H" & Mid$(HexInput, i, 2))  
  
Or alternatively just using the function I've shown in my first post here...  
  
Olaf

**Events**  
  
Closing()  
**Description**: Triggered when socket is about to close but has not yet been fully closed.  
  
Connect()  
**Description**: Triggered upon successful connection. Data may now be sent.  
  
ConnectionRequest(RequestID)  
**Description**: Triggered when a remote host is connecting to a listened port.  
  
DataArrival(BytesTotal)  
**Description**: Triggered when data has arrived in binary mode.  
  
Error(Number, Description, sCode, Source, HelpFile, HelpContext, CancelDisplay)  
**Description**: Triggered whenever an internal error occurs.  
  
**NOTE**: Unlike Winsock control, an error does not prevent operation and you are not required to close the connection.  
  
SendComplete()  
**Description**: Triggered when sending data has finished.  
  
SendProgress(BytesSent, BytesRemaining)  
**Description**: Triggered as sending data progresses.  
  
StatusChange(Status)  
**Description**: Triggered when status changes. However, error status is not received in favor of Error event.  
  
TextArrival(Text, LineChange, ANSI)  
**Description**: Triggered when a line of data has arrived. If text is Unicode, Text can be used directly. If text is ANSI, you must use StrConv or other means to process the text into an usable string.  
  
  
**Methods**  
  
Accept(RequestID) Boolean  
**Description**: Accepts a connection request from another UniSock control. The socket must be in a closed state before using this function. Upon successful accept Connect event is fired.  
  
**Result**: Returns True if the RequestID was valid.  
  
Bind([LocalPort], [LocalIP]) Boolean  
**Description**: Binds socket to currently set local port and ip. These may also be given as parameters. LocalIP can be an IP or a hostname.  
  
**Result**: Returns True if the IP and port were binded successfully.  
  
CloseSocket() Boolean  
**Description**: Cancels all asynchronous activity and closes the socket.  
  
**Result**: Returns True if the socket was closed.  
  
Connect([RemoteHost], [RemotePort]) Boolean  
**Description**: If protocol is TCP, begins connection to currently set remote host and port, or given host and/or port. If protocol is UDP, the current local ip and port are bind.  
  
**Result**: Returns True if the connection could be initialized (TCP) or True if the local ip and port could be bind (UDP).  
  
GetData(Data, [VarType], [MaxLen]) Boolean  
**Description**: Gets current incoming buffer to variable passed to Data. If passed variable is a variant, variable type set into optional VarType will be used. The default datatype is byte array. MaxLen sets the amount of bytes to retrieve. Whole buffer is retrieved if MaxLen is 0 or less, or greater than the buffer size.  
  
Unlike other Winsock control/class implementations, you may use all numeric datatypes and string: Boolean, Byte, Currency, Date, Double, Integer, Long, Single and String. All supported datatypes can be also be passed as arrays. String array separator is automatically detected (CRLF, LF, CR or NullChar). Strings are coerced from ANSI.  
  
**Result**: Returns True if the buffer was retrieved.  
  
Listen() Boolean  
**Description**: Binds currently set local ip and port if not bind already and starts listening for incoming connections. Upon connection ConnectionRequest event is fired.  
  
**Result**: Returns True if binding and initializing listening was successful.  
  
PeekData(Data, [VarType], [MaxLen]) Boolean  
**Description**: Identical to GetData, except the data is not removed from the buffer.  
  
SendData(Data) Boolean  
**Description**: Sends the given data to remote host. See GetData for supported datatypes. Strings are coerced to ANSI.  
  
**Result**: Returns True if inserting data to output buffer was successful.  
  
SendText(Text, [TextFormat]) Boolean  
**Description**: Sends a single line of string data. A line change character determined by LineChange is added to the string. TextFormat determines whether text is converted to UTF-8 or sent as is.  
  
**Result**: Returns True if inserting data to output buffer was successful.  
  
  
**Properties**  
  
BytesReceived() Long  
**Result**: Returns the current size of the incoming data buffer.  
  
LineChange() UniSockLineChange  
**Description**: Returns/sets the bytes that determine line change or array separator. This value is used when sending string arrays and when using SendText.  
  
LocalHostname() String  
**Description**: Returns the current hostname.  
  
LocalIP() String  
**Description**: Returns the current local ip.  
  
LocalPort() Long  
**Description**: Returns/sets the current local port.  
  
Mode() UniSockMode  
**Description**: Returns/sets the mode of the socket. Binary and text modes are supported. In binary mode DataArrival event is triggered. In text mode TextArrival event is triggered.  
  
Protocol() ProtocolConstants  
**Description**: Returns/sets the current protocol. TCP and UDP are supported.  
  
RemoteHost() String  
**Description**: Returns/sets the host to connect to, or returns the host that has been connected to.  
  
RemoteHostIP() String  
**Description**: Returns the remote host IP address as a string.  
  
RemoteIP() Long  
**Description**: Returns the remote host IP address as a 32-bit value.  
  
RemotePort() Long  
**Description**: Returns/sets the remote port to connect to, or port that has been connected to.  
  
RequestHost() String  
RequestIP() String  
RequestPort() Long  
**Description**: Information available about the remote host that makes a connection request to a listening socket. Use before accepting a connection to determinte whether you wish to accept the request.  
  
SocketHandle() Long  
**Result**: Returns the current open socket handle, or -1.  
  
State() StateConstants  
**Result**: Returns the current connection state of the control.

Okay, I know this is ambitious but this is the deal and it would be great if it were possible.  
  
Is it possible to use VB and Winsock to send raw packets on data as packets (real ones) and not just strings of data.  
  
Ie.  
  
0000 00 30 cd 00 07 f1 00 30 bd 64 97 2b 08 00 45 00 .0.....0.d.+..E.  
0010 00 28 4c af 40 00 80 06 c1 60 c0 a8 00 0c 50 e5 .(L.@....`....P.  
0020 db 26 06 ec 00 14 87 93 0e 0c 4a b0 3b b2 50 10 .&........J.;.P.  
0030 44 8b 5b 87 00 00 D.[...  
  
and then send them to an IP to be interpreted? (Considering that IP has the correct open port at that time)  
  
Jordok  
This is sending side..

VB Code:

1. Private Sub Command1\_Click()
2. Dim myArr(3) As Byte
3. myArr(0) = 97
4. myArr(1) = 98
5. myArr(2) = 99
6. myArr(3) = 100
7. Winsock1.RemoteHost = "127.0.0.1"
8. Winsock1.RemotePort = 12345
9. Winsock1.LocalPort = 12346
10. Winsock1.SendData myArr
11. End Sub
13. Private Sub Form\_Load()
14. Winsock1.Bind 12346
15. End Sub

and this is receiving Side

VB Code:

1. Private Sub Form\_Load()
2. Winsock1.LocalPort = 12345
3. Winsock1.Bind 12345
5. End Sub
7. Private Sub Winsock1\_DataArrival(ByVal bytesTotal As Long)
8. Dim myArr() As Byte
9. Winsock1.GetData myArr
10. For Each b In myArr
11. Text1.SelText = Chr(b)
13. Next

16. End Sub