Go to next page

Coping With Windows Communications

Use these undocumented tips, tricks, and work-arounds to customize your communications applications.

Carl Franklin is a Software Engineer at Crescent Software, 11 Bailey Ave, Ridgefield, Connecticut 06877. He is co-author of the ***Visual Basic Programmer's Journal*** Q&A column, and plays lead guitar in the VBITS Orchestra. Reach him at 203-438-5300, or on CompuServe 70662,2605. Crescent Software is a leading vendor of Visual Basic add-on products.

When you say the words "communications programming" or "serial-port programming" to Basic programmers they run screaming in fear. It is a fear of the unknown. Although we have always had the tools to communicate through a serial port, there is not much material available on the subject of serial-port programming for Visual Basic programmers. Fortunately, the concepts of communications programming are quite simple, and for the most part, a communications process uses only three or four functions. Relying on those functions, and some tips and tricks, you can do some simple serial-port tasks with Visual Basic for Windows Professional Edition, including dialing a phone number, monitoring a communications port for incoming data, writing it to a file, and more. Before we talk about programming, though, let's take a look at what serial communications programming lets you do, and the roles of the different players in the game.

Most personal computers come equipped with one or more serial ports. On the outside, a serial port is a connector on the back of the computer. If you have a mouse on your system, it might be plugged into a serial port. Described in simple terms, a serial port provides a channel for data to be sent and received by an external device.

At the heart of every serial port is a device called a universal asynchronous receiver transmitter, or UART (pronounced yoo-art). A UART accepts parallel data one byte at a time from the CPU and converts it into the equivalent stream of serial bits that is actually transmitted. The UART also receives serial bits and sends them to the CPU as byte values.

For serious communications developers, the best UART on the market is the National Semiconductor NS16550A. It contains an internal buffer, which takes a big load off the CPU when processing incoming data. If your computer's serial ports do not have 16550A UARTs, it is worth the $50 bucks to upgrade them.

Windows is a device-independent operating system, meaning that a Windows program will work pretty much in the same way from one machine to another, one video system to another, one printer to another, and one serial port hardware configuration to another. Windows accomplishes device independence by dividing the hardware-to-software connection in half. On the hardware side, the hardware vendor must provide a driver that connects its hardware to Windows. On the software side, an application need only call a generic routine to access the hardware. It is for this reason that Windows applications are device independent.

This idea is extended to serial ports. The Windows communications driver, COMMDEV, is the middleman in the communications process. A Windows application calls standard API routines for sending and receiving data, and the driver does the dirty work. Visual Basic is no exception. Because it uses the Windows API for communications, you can be sure your Visual Basic app will run on any Windows platform.

COMMUNICATIONS AS A VBX

When Microsoft first approached my company to help them develop the MSComm control, I must admit I was unsure that the world was ready to have the traditional method of serial communications programming turned on its ear. In the past, developers have had to use callable routines to manipulate serial ports, calling one routine to initialize the port, another routine to read data, another routine to send data, and so on. At first it was hard to grasp the benefit of wrapping these functions into a custom control, but then the merits became obvious. With such a control, one control equals one port. A comm port has settings, or properties, and it definitely has events that can occur. Thus, the custom control is the perfect model for a serial port.

Since the release of Visual Basic 3.0, Microsoft has released a newly updated version of MSComm that you simply must get. Significant improvements were made in its ability to work reliably in any situation. The file is called MSCOMM.EXE, and is available on CompuServe, in the MSBASIC forum, library 1. Several references are made in this article to the "latest version of MSComm." This is the version.

In a Visual Basic program, there are several preliminary steps you must take in order to get to the point where you can send a command to the modem. First, the serial port has to be opened. This is accomplished in three programming steps. Assuming that you have already placed an MSComm control on a blank form, you can add the following code to your Form1_Load procedure:

```
Comm1.CommPort = PortNum%
Comm1.Settings = SettingsString$
Comm1.PortOpen = True
```

The first line sets the number of the serial port. To use COM 1, PortNum% should equal 1; for COM 2, PortNum% should equal 2, and so on. Assuming that the mouse is using COM 1, our control should use COM 2:

```
Comm1.CommPort = 2
```

The second line sets the baud rate, parity, data-bit, and stop-bit settings for the comm port, in the form of "BBBBPD,S." The baud rate is the speed at which data travels through the port. This can be one of several set values. Most modems can support at least 2400 baud, and many support 9600 and 19,200. While some modems use compression methods to achieve higher baud rates, the highest baud rate Windows supports is 19,200 baud (higher speeds can be achieved with replacement comm drivers, however).

In serial communications, bits of data (ones and zeros) are represented as the state of a wire at a particular point in time. Since the timing is established by the baud rate, the smallest interval at which one bit of data can be received is clearly defined. At the next interval, if the state of the wire is off, the bit received is a zero. If the wire is on, the bit received is a one.

The parity method is a simple method of data validation. It is most common to not use parity checking as a means of data validation because it checks to see only if the number of bits received is an even or an odd number. There are other methods of parity checking that can be used, but for standard modem use, using no parity is most acceptable.

The data bits setting is the number of bits that are said to represent data. Once this number of bits has been received, the next bit or bits received are used as a flag to signify the end of the data. These flag bits are called the stop bits. In most cases, only one bit is used as a stop bit, because that is all that is necessary.

Most modem software uses a setting of N,8,1—no parity, eight data bits, and one stop bit. CompuServe uses a setting of "E,7,1." That is, even parity, seven data bits, and one stop bit.

For this example, we will set up the port at 2400 baud using no parity, eight data bits and one stop bit:

```
Comm1.Settings = "2400,N,8,1"
```

To open the port, simply set the PortOpen property to True. The completed example of opening a serial port looks like this:

```
Comm1.CommPort = 2
Comm1.Settings = "2400,N,8,1"
Comm1.PortOpen = True
```

The QuickBasic/Basic 7/VBDOS equivalent to the above code is:

```
FileNum = FREEFILE
OPEN "COM2:2400,N,8,1,BIN" FOR BINARY _
AS #FileNum
```

MANAGING INPUT OUTPUT AND BUFFERS

When a serial port is opened, The MSComm control creates a receive buffer and a transmit buffer. When one byte of data arrives at the comm port, Windows moves the byte into the receive buffer and tells MSComm that a byte has arrived. MSComm's job is to let the programmer know about it, which it does by incrementing its InBufferCount property by one (the InBufferCount property returns the number of bytes in the receive buffer).

You could monitor the number of bytes received simply by reading the value of the InBufferCount property in a loop:

```
Do
   Me.Caption = Str$(Comm1.InBufferCount)
   DoEvents
Loop
```

You can read data from the receive buffer into a string variable with MSComm's Input property. Here's how to read the contents of the receive buffer into the variable In$:

```
Comm1.InputLen = 0      'See below.
In$ = Comm1.Input       'Read the received data.
```

The InputLen property defines the number of characters that are to be read with the Input property. Setting InputLen to zero tells

MSComm to read all bytes in the receive buffer. You can read every received character and display it in the debug window like this:

```
Sub Form_Load ()
  Comm1.CommPort = 2                'Open and
            'initialize the port.
  Comm1.Settings = "2400,N,8,1"
  Comm1.PortOpen = True
  Comm1.InputLen = 1  'Read one
'character at a time.
   Do
     DoEvents              'Allow events.
     If Comm1.InBufferCount Then
'Received any data?
       In$ = Comm1.Input           'Read in one
character.
       Debug.Print In$;'Display in debug window
     End If
  Loop
End Sub
```

You must call DoEvents when processing communications in a loop. This allows for other processes to occur, including MSComm's handling of received data.

Based on this idea, take a look at Listing 1, a simple program that opens the comm port and writes all received data to a file until it receives an end-of-file character (decimal 26). After the port is opened, the program goes into a loop, constantly checking the InBufferCount property to see if data has been received. If so, the characters are read and immediately written to the open file. If the data received is a Chr$(26), the file is closed, the port is closed, and the program ends.

Just as the Input property reads data from the receive buffer, the Output property sends data into the transmit buffer, where Windows then sends it out the serial port. The transmit buffer is similar to the receive buffer. When you send a string out the serial port with MSComm's Output property, MSComm places the data in a transmit buffer, and tells Windows to send the data.

To monitor the number of characters in the output buffer, use the OutBufferCount property. The code will wait until one string has been sent before sending another string:

```
'Send a string
Comm1.Output = "This is the first _
            string." & Chr$(13)
'Wait until the entire string has been 'sent
Do
   DoEvents
Loop Until Comm1.OutBufferCount = 0
'Send another string
Comm1.Output = "This is the second _
            string." & Chr$(13)
```

When you open a port with the MSComm control, you are opening it in binary mode. That is, VB does not add or take away any characters, or filter out characters, unless you want it to. Consider the following QuickBasic command:

```
PRINT #1, "HELLO WORLD"
```

Assuming that file number 1 is an open serial port, this code will send the string "HELLO WORLD" followed by a carriage return out the port to the modem. MSComm's Output property sends a string exactly as you specify, like this VB code:

```
Comm1.Output = "HELLO WORLD"
```

Assuming that Comm1 is an MSComm control whose port is open, this code will send the string "HELLO WORLD" with no carriage return. If a carriage return is required, you have to specify it:

```
Comm1.Output = "HELLO WORLD" & Chr$(13)
```

Also, you can flush, or delete, the contents of both the receive buffer and transmit buffer by setting their InBufferCount property to zero.

Go to next page

Go to next page

Go to next page

Go to next page

THE MYSTERIOUS ONCOMM EVENT

MSComm's OnComm event is fired when a communications event or error occurs. Events include a change in the carrier detect line or other serial-port register, and a received character, for example. Errors include lost data due to a hardware overrun, receive buffer overflow and parity errors. You determine which of these caused the OnComm event to fire by reading the CommEvent property. This is explained in detail in the VB manual. However the manual does not give much detail regarding how OnComm is called and handled internally.

When you set RThreshold greater than zero, the OnComm event will fire with CommEvent equal to 2 (MSCOMM_EV_RECEIVE) whenever (RThreshold) number of characters are received. If new characters are received while the OnComm event code is executing, the executing code must finish before the next OnComm event will fire. The only way that you could allow OnComm to fire while code is executing is to call DoEvents in the OnComm event procedure.

What happens if you don't read the received data when a receive event occurs? The original version of MSComm and the current version of MSComm handle this behavior differently. In the original 3.0 release, when OnComm (EV_RECEIVE) is fired and you do not read the received character, you must wait for another character to be received before OnComm is called again. This proved to be difficult when reading one character at a time (InputLen=1). Because only one character can be read with each OnComm (EV_RECEIVE) if you miss reading one character, you will always be one character behind from what you expect.

In the current version, when OnComm (EV_RECEIVE) is fired and you do not read the received character, another OnComm (EV_RECEIVE) is fired until you have read it, or until there are no more characters in the receive buffer.

The WaitFor routine shown in Listing 2 is perfect for reading data from the receive buffer. You specify the string that you want to receive, and WaitFor doesn't come back until it receives that string. You must pass a timeout value in seconds, which tells WaitFor to come back if the string was not received in time.

What about when you need to receive variable-length strings? Let's look at some code that demonstrates reading variable-length strings from a machine. In this case, all strings end with a Chr$(0) or a Null character. The machine may send one character or it may send 100 characters, or any number in between. The only characteristic of the string is that it ends in a Null. The WaitFor routine waits for a Chr$(0) to be received, and returns:

```
'Assuming that the port is open
Z$=Chr$(0)
TO%=30
Do
  In$=WaitFor$(Comm1,Z$,TO%,_
                    Error%)
  If Error% Then Exit Do
  Select Case In$
    Case "End Of Transmission"
       Call Cleanup
       Exit Do
    Case "Sending Fax"
       Call ReceiveFax
  End Select
  DoEvents
Loop
```

COMMON PROBLEMS WITH MSCOMM

Here are solutions to the five most common problems VB programmers encounter using MSComm.

1. MSComm does not receive data.

One of the most widely reported problems with MSComm is that once a link is established to a device such as a remote modem, data can not be received. In other words, MSComm will send data, but the receive buffer never contains any data. The reason is that the default value of the RTSEnable property is False. For devices that use RTS/CTS handshaking, unless MSComm's RTSEnable property is set to True, data will not be received. When a device uses RTS/CTS handshaking, it looks at the value of the RTS line before sending data. If the line is low (False) it waits until RTS goes high (True). By setting RTSEnable to True, this problem is easily solved.

If you are trying to send data via modem, problems can arise if you do not pause for at least eight seconds or so after receiving a CONNECT string from the modem. If your modem does not use data compression, and you're connected to a modem that does use data compression, you will receive your CONNECT string eight seconds or so before the other modem does. This is because modems that use data compression send a packet identifying the data compression mode after connecting. It takes eight seconds for the modem to time out. If you do not pause, the other modem will become extremely confused. You can solve this problem using a pause routine that you can call to safely pause for a given number of seconds:

```
Sub Pause (Seconds!)
   StartTime! = Timer
   EndTime! = Seconds! + Timer
   Do
      DoEvents
   Loop Until Timer >= EndTime!
End Sub
```

If you are still having trouble, there may be

```
Sub Form_Load ()
     'Open and initialize the comm port
    Comm1.CommPort = 1
    Comm1.Settings = "9600,N,8,1"
    Comm1.PortOpen = True
    Comm1.InputLen = 1
      'Open a datafile
     Open "C:\DATAFILE.DAT" For Binary As #1
     ' Show the form
     Me.Show
      'This loop occurs in the background
     Do
          'Allow events.
          DoEvents
           ' Did we receive a character?
          If Comm1.InBufferCount Then
                'Yes, read it.
              In$ = Comm1.Input
                'End of file?
                If Asc(In$) = 26 Then
                     'Yes, close up and exit
                    Close #1
                   Comm1.PortOpen = False
                     Beep
                     End
                Else
                     'Write to the file
                    Put #1, , In$
                End if
          End If
    Loop
End Sub
```

Include Loops. When processing communications in a loop of any kind, you should call DoEvents inside the loop. This allows for other processes to occur (including MSComm's handling of received data). This program opens the comm port and writes all received data to a file until it receives an end of file character (decimal 26). After the port is opened, the program goes into a loop, constantly checking the InBufferCount property to see if data has been received.

Let's take a closer look at modems and discuss how to use them effectively. A modem is a device that attaches to the computer through the serial port. Once a channel is established between the computer and the modem, commands can be sent to the modem to tell it to perform such tasks as dialing a phone number or connecting to another modem.

In simple terms, one modem must *originate* a call and another must *answer* in order for two modems to connect. The PC that originates the call is often called the "remote" PC, and the PC that answers the call (and usually provides a service) is often referred to as the "host" PC. The host PC's modem is set up to answer the phone. When the phone rings, the host PC's modem picks up the phone and issues an answer carrier. The remote PC's modem calls the host with an originate carrier. The two tones merge, and the modems are connected. In this state, the carrier detect line is high. MSComm has a property called CDHolding, which returns True when the carrier detect line is high. As a general rule, if Comm1.CDHolding = True, then the modem is connected. The CDHolding property is supported by most modems.

Virtually all consumer-model modems sold today are Hayes compatible. In earlier days, the Hayes company had the best-selling modem. Subsequently, the Hayes command set has become the standard.

All Hayes modem commands start with the letters "AT." This stands for Attention and is used to wake up the modem, meaning that it identifies a string sent to the modem as being a modem command. The command to dial a phone number is "D." This is usually followed by "T" for touch tone phones, or a "P" for pulse (rotary) phones, and then the phone number. For example, just a few lines of code opens COM2 at 2400 baud, no parity, eight data bits, and one stop bit, and tells the modem to dial local information using touch tone:

```
Sub Command1_Click()
  Comm1.CommPort = 2
  Comm1.Settings = "2400,N,8,1"
  Comm1.PortOpen = True
  Out$ = "ATDT555-1212" & Chr$(13)
  Comm1.Output = Out$
End Sub
```

In this code:
"AT" means "Hey Mr. modem, I'm talking to you."
"D" means "Dial the following number."
"T" means "Using touch tone."
"555-1212" is the phone number (spaces and hyphens are OK).
Chr$(13) is a carriage return, which must follow all "AT" commands.

In this example, the modem is in "command mode." In other words, the modem is not connected to another modem, and is accepting modem commands. After you send the modem an AT command, it sends you what is called a "result code." If the modem successfully processed the AT command, the result code will usually consist of "OK" followed by a carriage return and a line feed. If you have ever used a terminal program, such as Procomm, DynaComm, QModem, or Windows TERMINAL.EXE, for that matter, you may have seen an "OK" on the screen. That is the modem telling you that it has successfully processed the last command it was given. To see if a modem is connected and properly initialized, send it "AT" & Chr$(13), and you should receive "OK" & Chr$(13) & Chr$(10).

Once the modem is connected to another modem, it is said to be in online mode. In this mode, everything you send to the port goes through the modem to the other side. If you are online and you wish to return to command mode, you can send what is called the escape sequence, which is a string usually consisting of three plus signs ("+++"). When you send this string across a modem line, the modem will drop back into command mode, where you can send it more AT commands, such as "ATH," the command to disconnect. Note that just because the modem is in command mode, it hasn't necessarily disconnected. From command mode, you could go back online by issuing the "ATO" command.

The sample code for dialing local information would be a fine example of a phone-dialing program, except that once the number is dialed, then what? The modem still has control of the phone line. The key here is to give the modem another command, the semicolon, which tells it to return to command mode after it has dialed the number. Once it has returned to command mode, the modem issues an "OK," and you can tell the modem to release control of the phone with the hang-up command ("ATH"), letting you converse with whomever picks up the phone on the other end. This phone dialer program hangs up after dialing:

```
Sub Command1_Click()
  '--- Open and initialize port
  Comm1.CommPort = 2
  Comm1.Settings = "2400,N,8,1"
  Comm1.PortOpen = True
  '--- The semicolon tells the modem to return
  '    to command mode after dialing.
  Out$ = "ATDT555-1212;" & Chr$(13)
  '--- Dial the number
  Comm1.Output = Out$
  '--- Input reads all data
  Comm1.InputLen = 0
  OK$ = "OK" & Chr$(13) & Chr$(10)
  '--- Wait until we get an "OK"
  Do
    DoEvents
    If Comm1.InBufferCount Then
      '--- Read new data
      In$ = In$ + Comm1.Input
      '--- Do we see an "OK"?
      If Instr(In$, OK$) Then
        Exit Do
      End If
    End If
  Loop
  '--- Send the hangup command.
  Comm1.Output = "ATH" & Chr$(13)
End Sub
```

While this code works very well, it assumes that you have the modem plugged into the wall jack, and the telephone plugged into the back of the modem. It also assumes that you have the telephone handset off the hook and are ready to start talking to the party on the other end. Obviously, if you delay picking up the phone until the modem tells you when the other party answers, you are out of luck because the modem disconnects immediately after dialing. ■

a more fundamental problem. MSComm is one of the only Windows communications products available commercially that uses the Windows 3.1 communications method called "Comm Notification." With this method, Windows notifies MSComm whenever a character is received. With the older and apparently more reliable Windows 3.0 method, MSComm must constantly check whether a character has been received.

After MSComm was released, it became apparent that no other major Windows communications software vendor was using the notification method because it was believed to have been somewhat unstable. In the current version of MSComm, there has been added a property called Notification, which determines whether MSComm uses the Windows 3.1 or the 3.0 method. The default setting is the more stable 3.0 method.

2. Out-of-stack-space errors occur.

This can happen if the RThreshold property is set to any value greater than one, and you have a call to DoEvents in your OnComm event procedure. Let's say that RThreshold is set to 1. This means that every time one character is received, the CommEvent property is set to 2 (MSCOMM_EV_RECEIVE), and the OnComm event is fired. Let's also say that in the OnComm event, you've coded a loop with a call to DoEvents somewhere in the middle of it. Whenever that DoEvents is called, there is an opportunity for another character to be processed, and the OnComm event to fire again.

If a character is received while the code is in the loop, the current code position will be saved in the stack memory area, of which there is a limited amount, and OnComm will be called again. This behavior, called recursion, can happen again and again until the stack memory is depleted, causing a stack overflow error. The cure is simple. If you must call DoEvents in the OnComm event, first set RThreshold to 0. This will ensure that OnComm will not be called if another character has been received. After you have called DoEvents, you can set RThreshold back to its original value.

3. RING event does not fire.

Unfortunately, some modems do not accurately reflect the status of the ring indicator (RI) line. An alternative and much safer method to determine if the phone is ringing when using a modem is to look for the word, "RING" to be sent by your modem. This occurs whenever the phone rings. Most commercial modem software relies on this method rather than checking the status of the UART's RI line.

4. Receive-buffer-overflow errors occur.

In the shipping version of MSComm, whenever a byte with a decimal value of 26 (the end-of-file character) is received, the next character received fires an RXOVER error via the OnComm event. This was in the original spec. The current version of MSComm does not include this feature.

5. EV_SEND event does not fire.

The send event requires a little explanation. At first glance, it looks like this event is fired whenever a character is sent out the comm port. This is not true. If you set SThreshold to 1, in order for EV_SEND to fire, there must first be two bytes (characters) in the transmit buffer, and then the next byte must be sent, leaving 1 character in the receive buffer.

As you can see, communications programming isn't that difficult once you understand the possibilities and problems. And although there are several common problems, the solutions are fairly straightforward once you know what to look for, and once you know the work-arounds. ■

WHEN USING LOOPS, YOU SHOULD CALL DOEVENTS INSIDE THE LOOP.

```
Function WaitFor$ (Comm1 As Control, Wait$, Timeout%, ErrCode%)

'Arguments:
'     Comm1 The comm control name
'     Wait$    The string to wait for
'     Timeout%    Number of seconds to wait
'Return values:
'     WaitFor$ returns all data that was received.
'     ErrCode returns True if a timeout occurred.
    '. InputLen must be 1
    OldLen% = Comm1.InputLen
    If OldLen% <> 1 Then
        Comm1.InputLen = 1
    End If
    'Initialize variables in case they were passed in.
    ErrCode = False
    Received$ = ""
    'And they're off!
    Start! = Timer
    EndTime! = Start! + WaitTime%
    Do
        'Most important
        DoEvents
        'Have we received any data?
        If Comm1.InBufferCount Then
            'Add it to In$
            In$ = In$ + Comm1.Input
            'Did we get what we're waiting for?
            There% = InStr(In$, Wait$)
            If There% Then
                'That's it, we're done
                Received$ = In$
                Exit Do
            End If
            'Is In$ getting too big?
            If Len(In$) > 4000 Then
                'Trim it
                In$ = Right$(In$, Len(Wait$))
            End If
            'Are we out of time
            If Timer >= EndTime! Then
                'Set the error code and exit
                ErrCode% = True
                Exit Do
            End If
    Loop
    'Reset InputLen
    If OldLen% <> 1 Then
        Comm1.InputLen = OldLen%
    End If
End Sub
```

Wait For The Data. Here's how to guarantee that you read data from the receive buffer. You specify the string you want to receive, and WaitFor doesn't come back until it receives that string. You must pass a timeout value in seconds, which tells WaitFor to come back if the string was not received in time.

AFTER MSCOMM WAS RELEASED, NO OTHER MAJOR WINDOWS COMMUNICATIONS SOFTWARE VENDOR WAS USING THE NOTIFICATION METHOD.

COMMUNICATIONS LIBRARY 2.1
A full-fledged communications DLL that enables you to access up to eight serial ports simultaneously. Includes XMODEM, YMODEM, YMODEM-G, ZMODEM and CompuServe B+ protocols for automatic file transfers. Includes a fully functional terminal program as an example. $149. SC: N/A. MicroHelp, Inc., 4359 Shallowford Industrial Parkway, Marietta, GA, 30066; 800-922-3383, 404-516-0899. Fax: 404-516-1099.

COMPRESSION PLUS
Compress Visual Basic arrays, blocks of memory, files, text and graphics screen images to a fraction of their original size. Includes full support for the ZIP file format. Includes an installation utility which you can use to distribute your applications in compressed format. Also available for Microsoft PDS 7.x, Basic 6.x, QuickBasic 4.00b+, and Visual Basic for MS-DOS. $149. SC: Yes. NC. ElfTech Development, Inc., 4374 Shallowford Industrial Parkway, Suite B, Marietta, GA, 30066; 800-553-1327, 404-928-8960. Fax: 404-924-2807.

CRYSTAL COMM FOR WINDOWS 3.23
A PC modem communications library designed for the Windows environment. The Communications Library provides the XMODEM, XMODEM CRC, XMODEM-1K, Tire, YMODEM, YMODEM Batch, ZMODEM, Streaming, KERMIT and ASCII communication protocols. $175. SC: Y/$350. Crystal Software, 329 Fire Lake Rd., Crystal Falls, MI, 49903; 906-822-7992. Fax: 906-822-7994.

DISTINCT TCP/IP FOR WINDOWS-SDK: VISUAL EDITION
Designed specifically for Visual Basic programmers. Contains custom controls for Windows Sockets, Telnet, and FTP, allowing developers to write TCP/IP applications without making DLL calls. Several samples in Visual Basic are also included. Can run over other Windows Sockets-compliant TCP/IP stacks, including Microsoft NT. $195 + $15 S&H. SC: N/A. Distinct Corp., 14395 Saratoga Ave., P.O. Box 3410, Saratoga, CA, 95070; 408-741-0781. Fax: 408-741-0795.

DYNACOMM
Full line of communication from asynchronous DEC connectivity to IBM 3270 connectivity. By midsummer with each DynaComm product, Visual Basic system users will be able to visually link their applications to DynaComm using DynaComm Custom Controls. Planned to support IBM, HP, NEC, and Data General mainframes. $249.95. SC: N/A. FutureSoft, 1001 South Dairy Ashford, Ste. 101, Houston, TX, 77077; 713-496-9400. Fax: 713-496-1090.

FAX PLUS
Include full incoming and outgoing fax support in your VB application using any Class 1, Class 2, Class 2.0 or CAS-compatible fax modem. Incoming faxes can be automatically saved as files or processed directly by your VB application. Outgoing faxes can be generated from graphics files or redirected from any Windows (TM) application that supports printing. $249. SC: N/A. ElfTech Development, Inc., 4374 Shallowford Industrial Parkway, Suite B, Marietta, GA, 30066; 800-553-1327, 404-928-8960. Fax: 404-924-2807.

OBJECT-FAX 3.0
A complete fax solution for Windows 3.1 and DOS. Supports OLE 2.0, DDE, drag-and-drop, Microsoft Mail, CCmail, Lotus Mail, TrueType, PostScript, PCL5, and automatic routing. Available in three different packages: Object-Fax 3.0 for large networks, Object-Fax Lite for small networks, and Object-Fax Single User: $; Call. Available Q1'94. SC: N/A. Traffic Software, 360 W. 31st St., New York, NY, 10001-2705; 212-714-1564. Fax: 212-714-1691.

QUICKLINE VERSION 1.3
A development library with macros to control the Multi-Line Voice Communications Board from Talking Technologies, Inc. Turns your telephone into a communications and marketing tool when hooked up with Visual Basic. Turn a PC into an answering system — rent voice-mail boxes, or take orders 24 hours a day. $500. SC: Call. Silicon Valley Products Corp., 18 Panuntuck Ave., P.O. Box 564, E. Moriches, NY, 11940-0564; Distributor: 800-447-9120. Fax: 516-878-1826.

PDQCOMM FOR WINDOWS
Crescent Software's enhanced version of MSComm. PDQComm can transfer binary files in the background using ZModem, YModem-G, YModem, XModem, CompuServe B+, or Kermit protocols, optionally displaying a status dialog box with a percent-complete meter. Can emulate TTY, ANSI, DEC VT100, and VT52 terminals without coding. PDQComm comes with ModemWare, a complete set of Visual Basic routines for handling modems, including a database of initialization strings for more than 440 modems. Currently PDQComm for Windows and MSComm are the only two communications custom controls on the market. C source code is available. No royalties. $149. Crescent Software, Inc., 11 Bailey Ave., Ridgefield, CT, 06877; 800-352-2742. Fax: 203-431-4626. ■