Using the eVB COMM control

Written by David Knechtges (davidknechtges@yahoo.com)
Edited by Derek (derek@deVBuzz.com)

This example demonstrates the use of the COMM control within embedded Visual Basic. Serial communications is fundamental to allowing a PC (or in this case a PDA) to talk to modems or other devices. Modems all generally use the Hayes "AT" command set for setup. This set of ASCII text-based commands allows the PC to initialize the modem, tell it to dial a number, hang-up, etc. Other devices use serial communications for diagnostics and troubleshooting purposes. For example, all cars made since the advent of computer based diagnostics have a diagnostic port. Generally these ports are located under the dash, the hood, within the fuse box, or any other number of places. While not RS-232 based, a conversion can be used to allow a PC to connect to the car's diagnostic port to allow troubleshooting and diagnosis. Unlike the modem commands, these serial buses all use binary communications. Basically, this means that all 256 possible characters are interpreted as their binary value and not their text value.

Embedded Visual Basic's Comm control supports both text-based and binary-based RS-232 communications. However, the documentation provided with eVB is incorrect when it comes to binary-based communications. This tutorial will walk through a text-based communications and a binary-based communications. It will demonstrate where the documentation doesn't work for binary-based communications, and what the work-around for binary-based communications is. Note, however, that under Visual Basic 6.0 for the PC, binary-based communications does work properly.

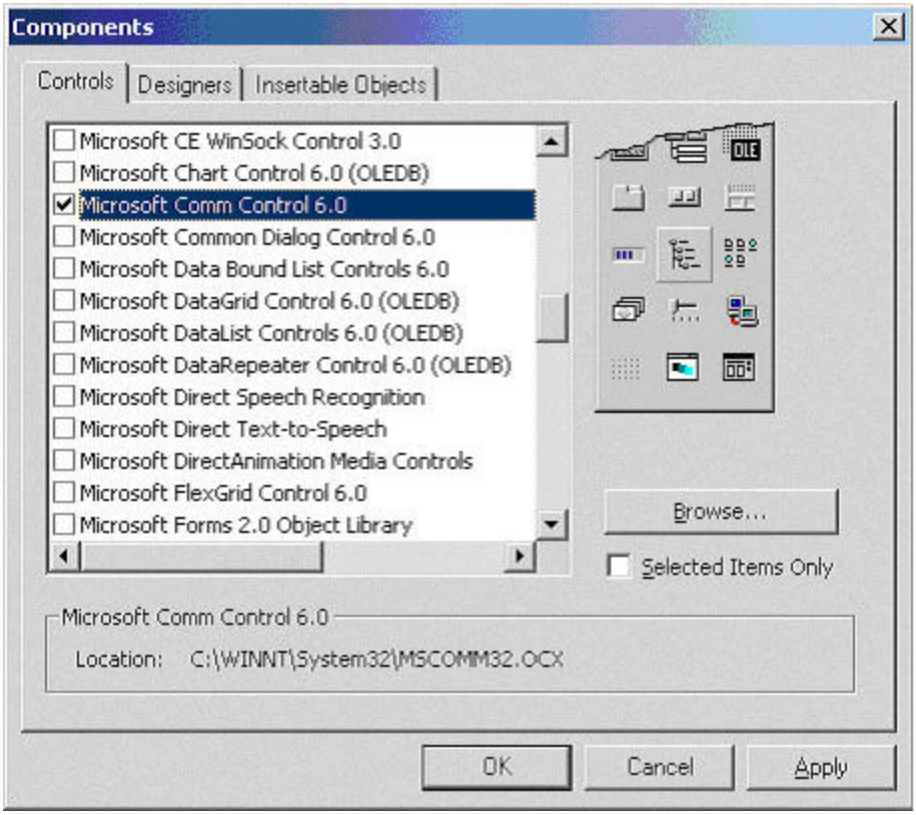
The set-up I am using is an iPaq with a CF sleeve and a CF serial card made by Socket. On my setup, this device uses COM4. You will need to determine what COM port your Pocket PC serial device uses for its communication. This tutorial requires that you have access to a PC with Visual Basic 6.0 installed on it. This allows the Pocket PC device to communicate with something. You will also need a null modem cable to connect the Pocket PC device to the PC.

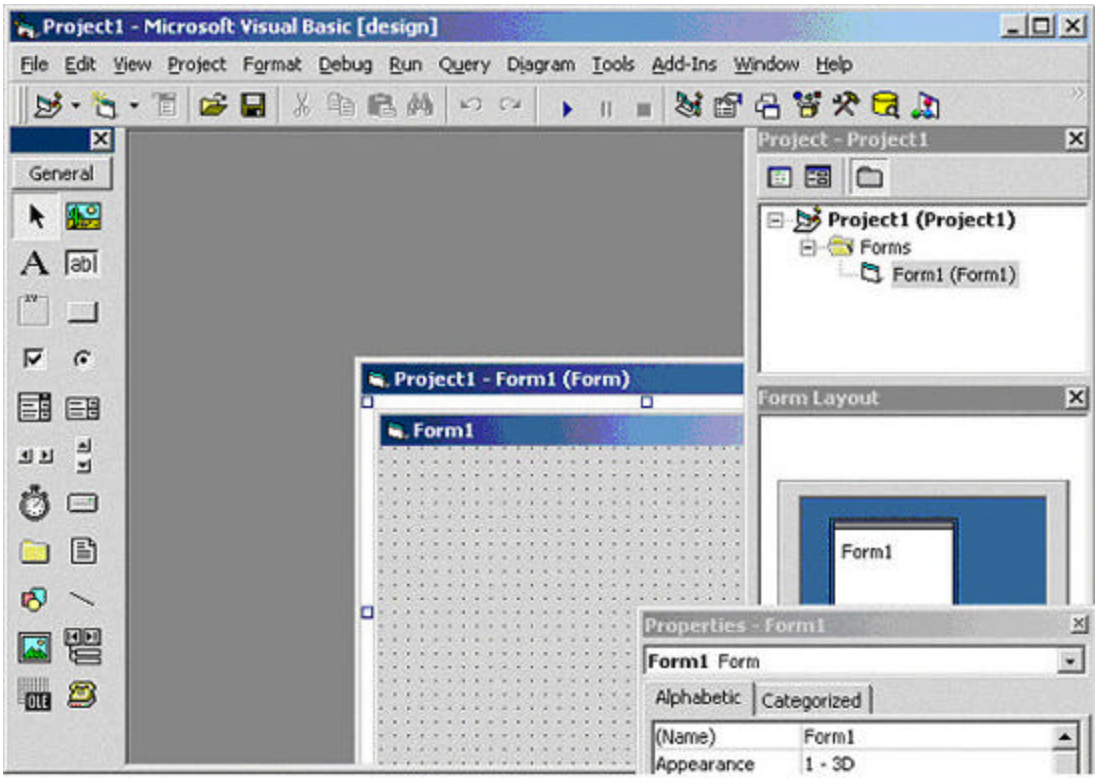
Because this can get a little confusing as to which item I am referring to, when I mention the PC, this is going to refer to the Visual Basic 6.0 application to talk to the Pocket PC. When I am mentioning the Pocket PC, this is going to refer to the eVB application.

Because it is much easier to simulate, I will extensively discuss text-based communications. Binary-based communications will be discussed as to how to make them work, along with a few code fragments to get you on your way.

Text-based Communications

Let's begin with text-based communications and we will begin with the PC side of things. Follow these steps in order:
1) Create a new project.
2) Go into the Project menu, select components, and add the Microsoft Comm Control 6.0 to the project. You should now see a telephone icon in the component window.



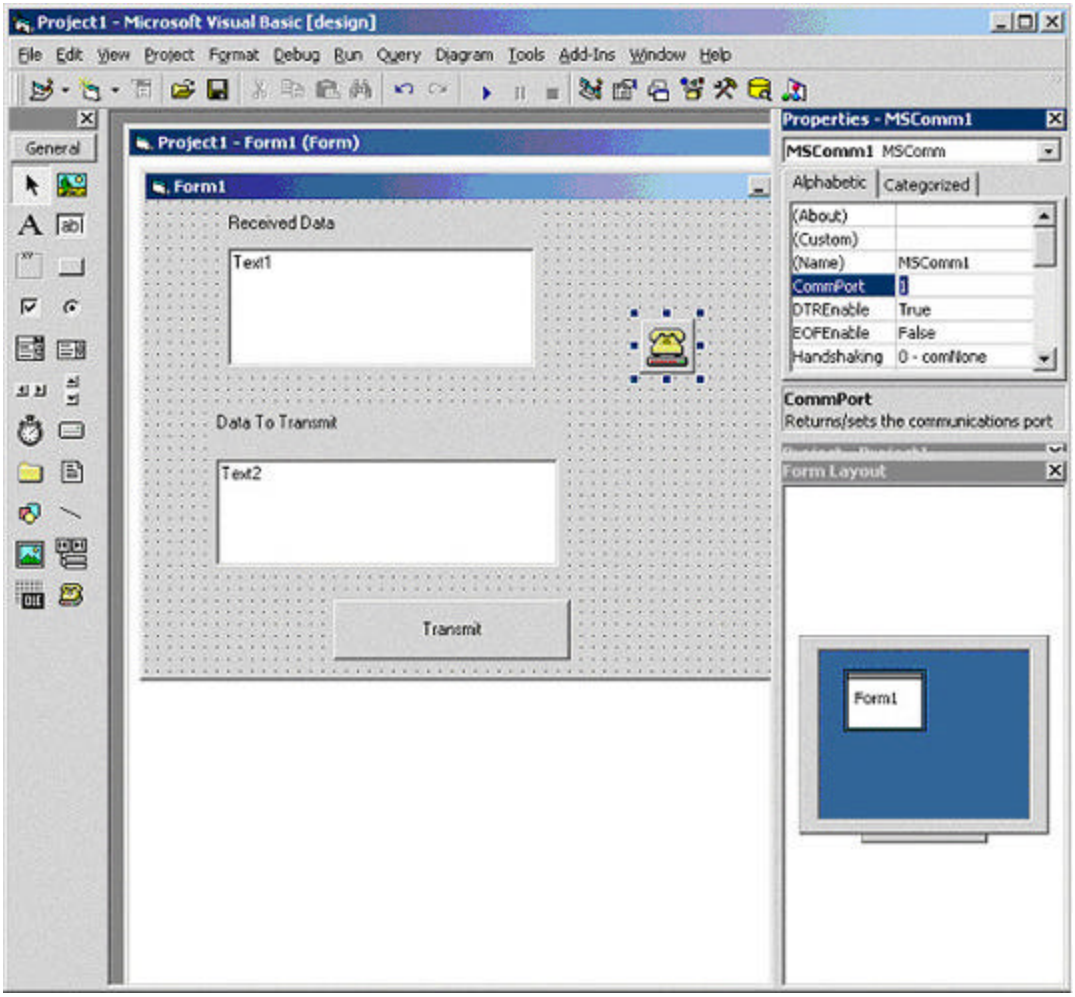


3) Place two text boxes on the form.
4) Place a label above each text box.
5) Change one of the label's captions to Received Data and the other to Data To Transmit.
6) Rename the text box below the Received Data label to ReceivedDataTextBox.
7) Rename the text box below the Data To Transmit label to DataToTransmitTextBox.
8) Create a command button on the form and change its caption to Transmit.
9) Select the Comm control and place it on the form.
10) Set the Comm Port property to the port you are using for communications on the PC.
11) Set the settings property to 19200,n,8,1.
12) Set the Rthreshold property to 1. This will cause an OnComm event any time a character is received on the serial port.
13) Fill in the form's code as shown:

```
Private Sub Command1_Click()
  MSComm1.Output = DataToTransmitTextBox.Text
End Sub

Private Sub Form_Load()
  MSComm1.PortOpen = True
End Sub
```

```
Private Sub MSComm1_OnComm()
  Select Case MSComm1.CommEvent
  Case comEvReceive
    ReceivedDataTextBox.Text = MSComm1.Input
  Case comEvSend
    ' do nothing here for now
  End Select
End Sub
```

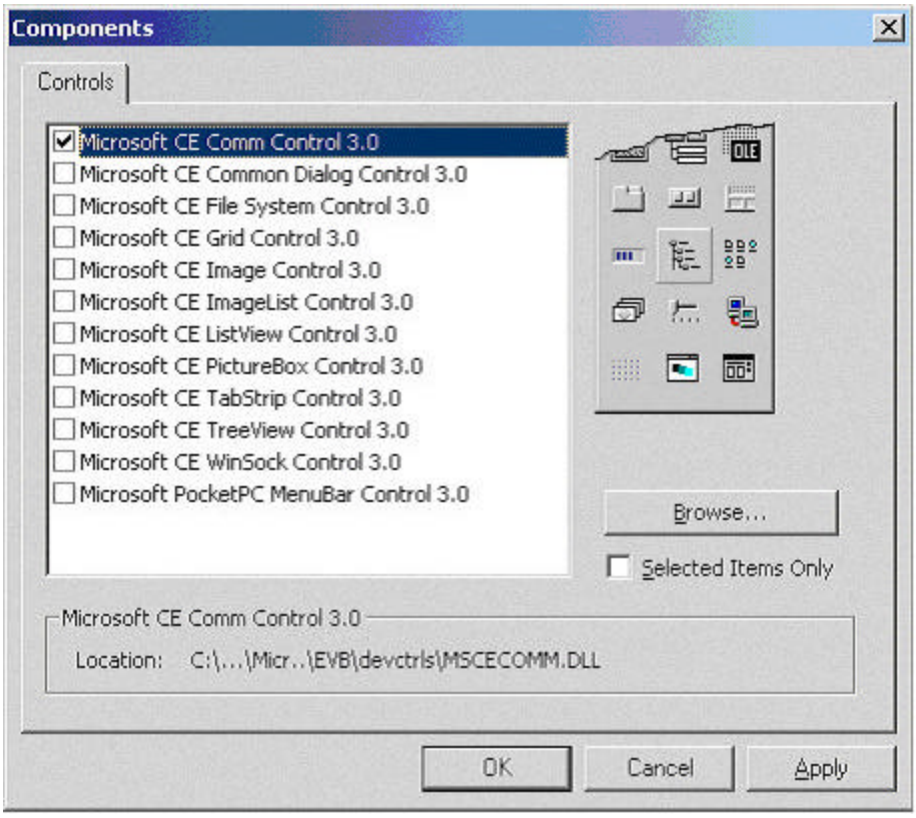
When all done, the PC project should look like:



We will now move to the Pocket PC side of things. Follow these steps for the Pocket PC application:

1) Create a new project.

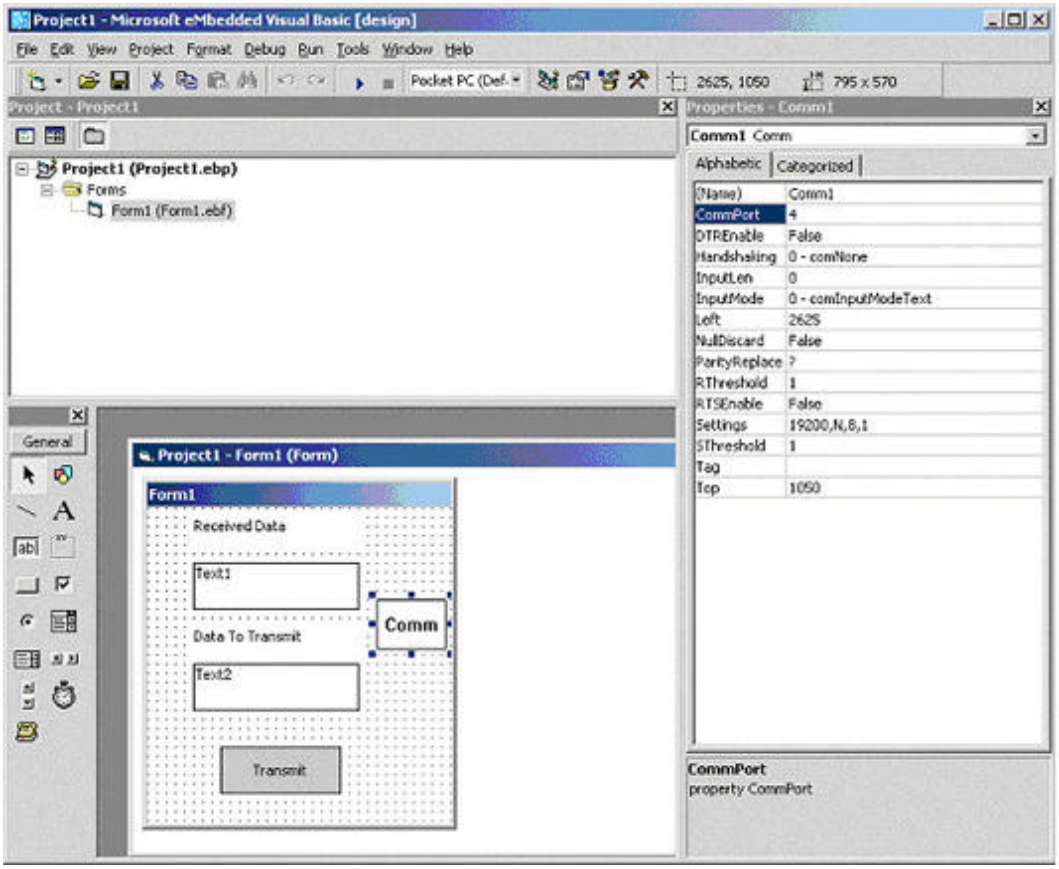
2) Go into the Project menu, select components, and add the Microsoft CE Comm Control 3.0 to the project. You should see a telephone icon in the components window.



3) Place two text boxes on the form.
4) Place a label above each text box.
5) Change one of the label's captions to Received Data and the other to Data To Transmit.
6) Rename the text box below the Received Data label to ReceivedDataTextBox.
7) Rename the text box below the Data To Transmit label to DataToTransmitTextBox.
8) Create a command button on the form and change its caption to Transmit.
9) Select the Comm control and place it on the form.
10) Set the Comm Port property to the port you are using for communications on the Pocket PC.
11) Set the settings property to 19200,n,8,1.
12) Set the Rthreshold property to 1. This will cause an OnComm event any time a character is received on the serial port.
13) Fill in the form's code as shown:

```
Option Explicit

Private Sub Comm1_OnComm()
  Select Case Comm1.CommEvent
  Case comEvReceive
    ReceivedDataTextBox.Text = Comm1.Input
  Case comEvSend
    ' do nothing here for now
  End Select
End Sub

Private Sub Command1_Click()
  Comm1.Output = DataToTransmitTextBox.Text
End Sub

Private Sub Form_Load()
  Comm1.PortOpen = True
End Sub

Private Sub Form_OKClick()
  App.End
End Sub
```

When all done, the Pocket PC project screen should look like this:



Now, run the PC program and the Pocket PC program. You will notice that data is being transferred successfully between the PC and the Pocket PC.

Try differing lengths of messages in the data to transmit box on both the PC and the Pocket PC sides and notice what happens. You will see that up to about 8 characters, the data is transmitted and received fine, but beyond that, it starts to clip off! This obviously is not the intended behavior. A little modification to the programs is necessary to prevent this behavior.

After looking at the Rthreshold property, you may be thinking that this is how to get the right lengths of data in. This would work for fixed-length messages. In other words, the program knows ahead of time how many characters it is going to receive in each message from the other device. Normally, however, this is not known and other methods must be employed. These can either be software or hardware based. Software based methods use the data that is being transferred to determine the start and end of a message. Hardware based methods can use either the clear to send/request to send lines (CTS/RTS) or the data terminal ready/data set ready lines (DTR/DSR) to indicate

the start and end of messages. Normally, however, the CTS/RTS is used for a receiving device to "hold off" the transmitter while it is processing incoming data.

While working on this tutorial, I discovered that the ComEvDSR event within the Pocket PC does not work properly. Basically, I was taking the programs as shown above, and added a timer to the PC program that toggled the DTR line on the PC side every 500 ms. (The code was written as: MSComm1.DTREnable = Not MSComm1.DTREnable). When looking at the events on the PocketPC, the ComEvDSR event was not occurring, but the ComEvCD event was. This is the carrier detect event. If I did this exact thing in reverse, i.e. the Pocket PC side toggling DTR, and looking for ComEvDSR on the PC side, it worked properly. In other words, I am saying to use the DTR/DSR functionality at your own peril!

So, back to the original problem.. We can get the messages in properly using a form of CTS/RTS for our application. Let's first get the programs coded and running, then I will explain what these changes are doing.

For the PC side, add a timer to your form, set the Enabled property to False, and set the Interval property to 60. Select the MSComm control on the form and set its Sthreshold property to 1. This property will cause the comEvSend event to fire when the transmit buffer is empty. Now some new code will be added to the form. The code for the form should now look like this:

```
Dim InputData As String
Private Sub Command1_Click()
  MSComm1.Output = DataToTransmitTextBox.Text
End Sub
```

```
Private Sub Form_Load()
  InputData = ""
  MSComm1.PortOpen = True
End Sub

Private Sub MSComm1_OnComm()
  Select Case MSComm1.CommEvent
  Case comEvReceive
   InputData = InputData + MSComm1.Input
  Case comEvSend
    Timer1.Enabled = True
  Case comEvCTS
    ReceivedDataTextBox.Text = InputData
    InputData = ""
  End Select
End Sub
```

```
Private Sub Timer1_Timer()
  Timer1.Enabled = False
  ' all data sent, so toggle rts.
  MSComm1.RTSEnable = Not MSComm1.RTSEnable
End Sub
```

On the Pocket PC side, change the Sthreshold property in the Comm control to 1. Next, we will be changing the code. The code for the form on the Pocket PC side should now look like this:

```
Option Explicit
Dim InputData As String
Private Sub Comm1_OnComm()
  Select Case Comm1.CommEvent
  Case comEvReceive
    InputData = InputData + Comm1.Input
  Case comEvSend
    ' all data sent, so toggle RTS
    Comm1.RTSEnable = Not Comm1.RTSEnable
  Case comEvCTS
    ReceivedDataTextBox.Text = InputData
    InputData = ""
  End Select
End Sub

Private Sub Command1_Click()
  Comm1.Output = DataToTransmitTextBox.Text
End Sub

Private Sub Form_Load()
  InputData = ""
  Comm1.PortOpen = True
End Sub

Private Sub Form_OKClick()
  App.End
End Sub
```

Go ahead and run both programs. You should now see that anything you type on either side gets transferred properly to the other side.

Now for an explanation of what the code is doing... When a message has completed transmission, the side transmitting toggles the RTS line thereby causing a CTS event on the other side (remember that RTS and CTS are swapped in a null modem cable). This signal indicates that a message is complete, and allows the other side to begin processing it.A question you may be asking is why the code is different between the two

sides to get this to work. Feel free to code them exactly the same on both sides (getting rid of the timer on the PC side) and see what the results are. Basically, what is happening is that the PC is waiting about 60 milliseconds after its transmit buffer is empty before asserting the RTS line. The Pocket PC is not doing that. If you do not wait the period of time on the PC side before asserting the RTS line, you will notice the Pocket PC side delaying by one transmit display of its data. For example, the first time on the PC you try to transmit ABC, the Pocket PC will be blank in its received data text box. The second time you try to transmit DEF, the Pocket PC will display ABC. I chose 60 milliseconds because the Windows event timer on the PC is a 55 millisecond timer. Feel free to decrease the timeout on the timer downward and see the results. It does get very interesting down around 15-25 milliseconds. I may be wrong, but I believe the reason this happens on the Pocket PC side is that it does not finish receiving its data before it gets the CTS event and then fires it anyway. The PC side, however, does not exhibit this same behavior. It finishes receiving its data and then fires the CTS event as you would expect.

If you wanted to, you could make the code the same on both sides, just use a timer on both sides, and things should work out just fine.

This basically concludes text-based communications. Now for binary communications...

Binary Communications

The Comm control on the Pocket PC supports binary communications. NOTE: The documentation provided with eVB is wrong! I ran into many problems getting this to work and decided to share how I solved this problem. Binary communications using the Comm control under eVB must be done using text-based communications and "faking out" the control, so to speak. I found a solution under MSDN on Visual Basic 4.0 for the PC. I decided to try it out on eVB, and lo and behold, it worked. Set up the Comm control as though you were using text-based communications, i.e. set the Comm control's InputMode property to comInputModeText. Set the Comm control's Rthreshold property to 1. Set the Comm control's InputLen property to 0.

The following code fragment describes how to transmit a byte:

```
Dim TransmitByte as Byte
MSComm1.Output = Chr(TransmitByte)
```

The following code fragment describes how to receive a byte stream of up to 100 bytes:

```
Dim TmpStr As String
Dim StrLen As Long, I As Long
Dim FileData(100) As Byte
```

```
While MSComm1.InBufferCount > 0
  TmpStr = MSComm1.Input
  StrLen = Len(TmpStr)
  For I = 1 To StrLen
    FileData(I) = CByte(Asc(Mid(TmpStr, I, 1)))
  Next I
Wend
```

A description of what is shown above is probably necessary now. On the transmission side, we are taking the byte to be transmitted, converting it into a character (through the Chr function), and writing it to the Output property of the Comm control. Easy enough. Just a simple conversion is all that is needed. The reception side is not as clean, though. First, the "text" data is read from the input property of the Comm control into the TmpStr string. Then, using a loop to walk through the entire string, each character is extracted from the string using the Mid function, converted to ASCII through the Asc function, and then converted to Byte through the Cbyte function. Then the Byte value is written to the FileData byte array. The while loop checking the InBufferCount property is necessary to get in any more bytes in the byte stream that may appear while we are walking through the previous set of data.

Take the code fragments above and experiment.. I used this binary based communications successfully on a project I am currently working on that takes the Pocket PC and allows it to communicate with a serial device for configuring and troubleshooting some of my company's products.

Hopefully you now have an idea of how the Comm control works using a real application and will be able to do some of your own communications!