Introduction

I have worked for over 35 years as an engineer, and for the last 22 years I have worked designing modems and data communications systems hardware and software. I have also designed data acquisition and other electronic hardware and software. In almost every case, an integral part of the designs of these other systems was a data communications subsystem that frequently used RS-232 or similar serial methods.

I started designing software for Windows at the same time that VB 2.0 was released. I had looked at VB 1.0 but felt that it was not quite "ready for prime time," although there had been a lot of useful software written using VB 1.0.

As soon as I started using VB 2.0, I became a regular visitor to the CompuServe MSBASIC forum. I found the MSBASIC forum to be a valuable source for answers to questions and I quickly became familiar with the expert programmers that frequented the forum. When I felt competent to contribute to the online discourse, I did so. An area that was insufficiently covered was serial communications and, more specifically, the MSCOMM.VBX. Fortunately I already had a background in serial communications and modems (which generate more questions than any other subject in the area of serial communications). So, I started writing example code and answering questions online. I was satisfied that my efforts were beneficial; many correspondents sent me messages of thanks and other positive feedback. I also felt that I would like to do more. Microsoft has moved their developer tools onward with the release of the various incarnations of Visual Studio .NET. While serial communications may appear to be less important with the emphasis on networking and the Internet, it still has many important applications.

In early 1994, the Microsoft engineers in charge of the MSBASIC forum activity recognized my online assistance by naming me a Microsoft Developer MVP (Most Valuable Professional). The MVP program was started by Microsoft as a way to recognize those people who provided accurate and valuable online help to other developers. The MVP program was a way for Microsoft to say "Thanks" and to encourage future contributions from the MVPs. See http://mvp.support.microsoft.com/default.aspx for more information

about the Microsoft MVP program.

I have continued my participation in the Microsoft MVP program with Microsoft's move to the Internet.

So, Why Did I Write This Book?

I found that I could never spend as much time online as I wanted. And, I could not put as much detail into responses as I would have liked. Frequently, a simple response to a question brings up many subsequent questions. One of the most frequent questions asked of me was, "What book can be used to learn what you have told me?" There was none that I could find. I do not think that I can kill all questions with this one tome (puns and mixed metaphors are a favorite pastime of mine) but it gives me a chance to get everything down in writing that I think may be useful. It also has kept me out of my wife's hair for several months.

A common complaint that I have seen is that the documentation on MSCOMM that is furnished with Visual Basic is incomplete and borders on useless. Well, let me be charitable and say that it is not as practical as I would like, that it does not cover most of the issues that I'm tackling here. I also am able to furnish information on commercial serial communications products.

When Visual Studio 2002 and 2003 were developed, serial communications support was **not included.** Thus, resources were even more limited.

I have had plenty of help gathering information and resources for this book. Please see the Acknowledgments section for a list of the people who have helped me out. If I make an error conveying to you, either in fact or by implication, something that they have provided to me, please let me know and I will try to rectify the problem.

Please help me out by letting me know when you find something that should be changed or included.

What Is In This Book

The working title for this book was *An Engineer's Guide to Practical Serial Communications in Visual Basic*. The title has been changed but the approach remains the same. I have tackled the problem of serial communications as an engineer would. That is, I have included background information on serial communications and details on how the various parts of a serial communications system act and interact. This "systems approach" should be useful to engineers and non-engineers alike.

I have tried to include detailed information on the use of the Windows API (Application Programming Interface) for serial communications, the use of the MSCOMM custom controls that are furnished with VB Pro and Enterprise versions and Visual Studio.NET, and commercial serial communications add-ons. In addition to general-purpose serial communications, I have included special purpose information on Alphanumeric Paging and a variety of other specialized serial communications topics.

WindowsCE devices are growing in popularity and utility and they present some unique problems that need to be addressed.

I have expanded the discussion of checksum and CRC calculation and have included working example code.

New to the Fourth Edition is a discussion of various Virtual Serial port implementations, including how some might be used by Visual Basic Programmers.

I have placed the discussion of XMCommCRC.ocx and associated example programs in a separate chapter. XMCommCRC.ocx is an ActiveX control that I wrote using VB6. XMComm wraps the functionality of MSComm32.ocx and adds XMODEM/Checksum and XMODEM/CRC error-corrected file transfer capability. It may be used freely in Visual Basic, Excel, Access, Visio, LabView, or other environments that support ActiveX controls. There is no license restriction on its use. The source code for XMCommCRC.ocx is the starting point for this new chapter. There I discuss its design and implementation. The accompanying examples illustrate the use of the XMCommCRC control.

I have devoted one full chapter to VB.NET, the Visual Basic portion of Visual Studio.NET. VB.NET examples cover the same general subjects that are in other chapters for earlier versions of Visual Basic. Several of these examples used the Upgrade Wizard that is part of the VB.NET development environment. The code generated by the Upgrade Wizard may not be as intuitive as one might hope plus it sometimes needs modification to work well. However, the Wizard does a fair job. These examples allow me to illustrate some of the built-in functions that the .NET framework offers to make a developer's life more productive. This productivity comes at a cost — I discuss that, too.

With the release of Visual Studio 2005 serial communications is included for the first time. The System.IO.Ports namespace provides this support. I have added detailed information on the use of this namespace, along with example programs that provide practical starting points for its use. I developed a native VB .NET class that encapsulated the Windows communications APIs and the FileStream and other .NET methods for use with Visual Studio 2002 and Visual Studio 2003. This class was in the 3rd Edition, and still is valid for those development environments, which had no built-in serial support. However, to make space for Visual Studio 2005 in this edition, I have moved the associated text to the CD ROM.

The 3rd Edition of this book included a chapter covering serial communications using eVB for Windows CE and the Pocket PC. However, with the release of Visual Studio 2003, Microsoft moved practical application development to the Compact Framework. The Compact Framework, like the desktop .NET Framework, did not include serial communications, so it was left to developers like me to create that functionality. The eVB chapter has been replaced by a Compact Framework chapter in the Fourth Edition. All of the text in the 3rd Edition Chapter 8 is retained on the CD ROM.

Occasionally, I write C# code. Some C# will creep into the examples on the CD ROM. While I'd like to cover C# programming in more detail, I will have to leave that to another time.

Included on the CD ROM are code examples to illustrate as many of the things that I talk about as possible. To use this example code, you will need to have installed the version of Visual Basic that is specified along with any custom controls that are mentioned. But, often, the example code will be portable to other versions of VB and other custom controls. The example programs are designed as applets. That is, each stands alone and implements one or more features that are discussed in the book. None of the applets depends on another, although there is, naturally enough, some overlap in details.

I have included a folder on the CD ROM for X10 communications. These controls, examples, and information are freeware, downloaded from the Internet. I do not have the space to go into any detail on this subject in the book.

The CD ROM that accompanies the book includes custom controls and products from a variety of vendors. Please see the documentation that accompanies each of these sets of software for ordering or registration information. Some of these vendors will offer readers of this book discounted software or other benefits. See the README or similar file in each associated folder on the CD ROM for details on these offers.

I have used this icon  **in the text where I want you to refer to the CD ROM for a complete program listing or for information and services directly from the publisher of the product being discussed.**

Also included on the CD ROM are several freeware DLLs and utility programs.

I have included as much information on debugging serial communications problems, both hardware and software, as I can. Debugging is critical and requires an understanding of the interaction of several disparate elements. These elements are the software that you have designed, the computers used and any multitasking that they must do, and any modems that are used. The tools that I mention are invaluable but there is no substitute for uncommon sense and for a methodical approach.

An appendix is devoted to resources. There are addresses, both mail and online (if available) and phone and fax numbers for all of the companies that are mentioned. A short description of the products that they offer, that may be valuable, is included. I have included as many other products as I could find, even if I do not discuss them explicitly in the book. If any product that should be included is not, let me know.

OK, What Is Not In The Book?

I have included nothing on serial communications via the Internet. This is a subject that should have a book devoted to it. There are several such books available and there will be many more coming. I have also decided NOT to include the use of fax add-ons or Microsoft Exchange. Time and resources have left this as a subject for the future.

I have included information on the features that are included with the MSCOMM32.OCX that is furnished with VB 5/6 and now VB .NET. However, some of the new features of VB .NET (and there are many that will be useful) will not be discussed in detail.

I do not try to cover synchronous serial communications. Windows has no built-in support for synchronous communications while Windows support for asynchronous communications is substantial. Synchronous hardware is uncommon but it is available. So, if this is your need, you will have to do some research or get some consultation in this very specialized area. However, one product that is included in Chapter 7 (see LUCA) includes synchronous communications support.

I do not cover the control of PLCs (Programmable Logic Controllers). It is possible to implement the more common PLC communications protocols using MSCOMM or some other add-on. But perhaps a more practical solution is to use one of the custom controls designed specifically for this purpose. I have listed some of them in the Resources appendix. Also, refer to the LUCA product description in Chapter 7 and on the CD ROM. LUCA includes support for several PLC related protocols. At the time of this writing, I had plans to include a variety of commercial add-ons that support PLC communications in the Resources portion of the CD ROM.

The Universal Serial Bus (USB) is not discussed. USB, as its name suggests, is a bus designed to provide "Plug and Play" capability to peripheral devices. It requires a device driver for supported devices on the bus, similar to printer, network and video adapters, or other conventional PC hardware that may be used. Unlike conventional hardware buses like PCI, USB devices are "daisy-chained" together using special serial cables. So, while the name says "serial", that is a physical description of the way that it connects. In other practical ways, the "bus" part of the name is what is important while serial is not. There are USB serial devices, however. These provide a way to add serial ports to a PC without opening the box and inserting a card. Use of these serial ports is the same as those on an internal card (with minor caveats that will be discussed in the Debugging chapter). Appendix A provides a list of pertinent vendors. Jan Axelson's book *USB Complete Third Edition, Everything You Need to Develop Custom USB Peripherals* (ISBN 1931448027) is a good source of information on USB. It has example code written in Visual Basic and VB .NET.

Bluetooth is a serial communications technology that might be of interest. I do not attempt to cover this in detail. Why not? First, often it is designed for networking computers and compatible devices. It may not be oriented toward raw serial communications but may use an extensive protocol stack; this depends on the actual Bluetooth device with which you are communicating. There are some Bluetooth devices that may be treated as standard serial ports (an example is most GPS receivers that provide a Bluetooth interface). Second, in general, Bluetooth will not use a conventional serial port, though there are serial to Bluetooth adapters. See the Resources Appendix for Bluetooth resources. Last, I have not had the need to develop any Bluetooth enabled applications, so there are more things that I do not know, than there are things that I do know about it. There is an old saying, "Those who can, do. Those who cannot, teach." While there is some small truth in this statement, I prefer to admit the limitation.

Most of the discussions in the book and all of the code samples on the CD ROM use the North American (English) versions of Windows and modems designed for North American use. Users in other areas of the world may find differences that invalidate one or more things that are covered. Unfortunately, this is a fact of life. Microsoft TAPI attempts to alleviate this problem and to date it has been only somewhat successful.

Parts of earlier Editions of the book have been moved to the CD ROM to make space for new text. These cover a variety of topics that may still be of use, such as the 16-bit Windows serial communications API, eVB, and various communications products that may still be present in legacy systems. Refer to the CD ROM 2ndEdition and 3rdEdition folders for .doc files that contain the deleted text.

If I learn something new and useful or if you let me know of something that I can mention, specifically, about serial communications in other parts of the world, I will add it to the next revision of this book. An important aspect of this book will be the continuous upgrades and improvements to the text and software.

How to Use This Book

This is not a "Dummies" book. I have no quarrel with that highly successful genre. But I do not think that serial communications is a subject for novices. I have assumed that you know how to program in Visual Basic. I do not go into how to edit code, how to place and use standard VB controls, or any details of UI (User Interface) design, except where those affect the subject at hand, serial communications.

I do not attempt to teach you a programming style or to use programming techniques that are specifically designed to encourage code reusability. If you want to learn how to program, how to design class modules, or to design a User Interface, see the Resources appendix. There are several books listed there that cover these subjects.

I do not assume that you have done any serial communications programming or that you have an intimate familiarity with modems or other serial communications devices. I will ask you to read the manuals that are furnished with the communications device that you are going to use. As painful as the fact may be, there is no better source for accurate information than that furnished by the manufacturer.

I will offer opinions on programming techniques and approaches to problem solving. These opinions are my own and contrary opinions may well exist. I will offer them to you because I have found them to be valuable to me. See Conventions and Style for more information.

A Quick Preview of the Book and CD ROM Content

Chapter 1 covers serial communications, Windows, the PC hardware that is used, and flow-control. Error-corrected file transfers and terminal emulations are covered next.

Chapter 2 covers modems and serial standards. More than 40 of the most common questions about modems are answered.

Serial hardware standards are discussed. Various RS-232 null-modems are illustrated. RS-422 and RS-485 are discussed, with code fragments presented for RS-485.

I have added a new section to this chapter for a more detailed discussion of checksum and CRC calculation in Visual Basic. Several examples are included.

Chapter 3 discusses the details of the Windows communications API, including the Telephony API.

Chapter 4 discusses the MSCOMM custom control. The versions for VB 2.0 through VB 6.0 are covered along with programming concepts for VB6. Examples for earlier versions of Visual Basic will be found on the CD ROM. This chapter is where we start to see some practical program examples to illustrate the use of MSCOMM.

There is a program included called DMM/LOGGER. This program illustrates how to use MSCOMM to communicate with a (DVM) Digital Volt Meter. The ideas that are conveyed with this program include how to implement a communications protocol. In this case, the protocol has been designed by an electronic instrument manufacturer.

Two MSCOMM programs implement Global Positioning Satellite receiver interfaces. These illustrate implementation of a serial protocol called NMEA-0183. One receiver program synchronizes your computer to the GPS receiver time and date and provides highly accurate location (latitude and longitude). The other decodes more NMEA data such as altitude, speed over the ground and the number of satellites in view but does not synchronize computer time.

A simple magstripe and barcode programs each are provided to illustrate the use of magstripe, barcode and similar scanning devices.

Several other programs (applets) are included to illustrate various points. See the CD ROM for example programs that did not make it into the text. Chapter 4 also covers XMCommCRC.ocx. XMComm is an ActiveX control written in Visual Basic 6.0 that adds XMODEM/checksum **and** CRC error checked file transfers to the underlying capability of MSComm32.ocx. It can be used in a variety of development environments that support ActiveX controls (the only exception is browsers where it is not appropriate to host an ActiveX control that accesses client-side hardware such as the serial port). The source code for XMComm illustrates the design of the ActiveX control. The XMTerm, Remote/Host and Flashlite example code illustrate the use of XMComm in actual VB applications.

Chapter 5 explores some of the new .NET territory. I provide four example programs that illustrate the opportunities and issues in Visual Basic.NET. I have ported three VB6 programs to .NET. One is the XMTermNET program which uses the XMCommCRC.ocx ActiveX control for terminal emulation and error-checked file transfer. The second is a GPS program that uses native .NET methods for converting UTC (Universal Coordinated Time) to local time and date information, and which automatically compensates for Daylight Savings time. The third program is a straightforward port of VBTerm, using MSComm32.ocx, to VB.NET. These programs illustrate the use of .NET COM Interoperability. That is, Visual Basic .NET and other .NET framework languages allow you to continue to use ActiveX controls and other ActiveX objects in .NET programs.

I have included one native VB.NET program that illustrates the .NET built-in support for serial communications using the FileStream class. Unfortunately, FileStream does not furnish all of the "hooks" needed for a complete implementation of a serial communications object. We have to use unmanaged code from the Windows communications API to do some of the heavy lifting. Thus, some of this code will look quite a bit like some of the equivalent code in Chapter 6. However, .NET does offer some real power that I mention. Method overloading is something new to VB programmers. It provides an important tool that we will use.

MS.NET is an evolving program for Microsoft. Developers will have to add new techniques to their toolset in order to work in this area. Visual Studio 2005 adds the System.IO.Ports namespace. This furnishes a native serial communications class. I discuss this and provide examples, including XMComNET (XMODEM for .NET).

Chapter 6 goes into the Windows 32-bit serial communications API. 16-bit API information and examples have been moved to the CD ROM.

Next is TAPI (the Telephony API). A simple Windows 95/98 dialer is shown plus other uses and limitations of TAPI are discussed.

Chapter 7 is an overview of a variety of commercial communications add-ons for Visual Basic. Sax Comm Objects, Greenleaf CommX, MagnaCarta CommTools, LUCA, amComm, SuperComm, and Crystal CrystalCOMM are discussed.

One example program is provided. This program uses Sax Comm Objects to create a program that provides a combination Host (BBS) and Remote (client) communications program that implements automatic file transfers. This program illustrates TAPI modem configuration, dialing, post-connection user validation, and automatic file transfers using the Zmodem file transfer protocol.

Previous editions of this book provided example programs that used PDQComm from Progress/Crescent Software. PDQComm is no longer available. However, the example code and accompanying text has been placed on the CD ROM in the PDQComm folder.

Chapter 8 discusses paging. Numeric paging is shown and its limitations are discussed.

Alphanumeric paging (AlphaPaging) is the real forte of this chapter. One commercial Visual Basic add-on is discussed: the Logisoft Page/X ActiveX control.

A program is included that uses Ron Tanner's PowerPage DLLs (the DLLs are included on the CD ROM) and illustrates the AlphaPaging process. This DLL is no longer available, so this has been moved to the CD ROM.

Chapter 9 covers serial communications using VB .NET and the Visual Studio 2003 Compact Framework 1.x. Serial Programming for Visual Studio 2005 and the Compact Framework 2 is equivalent to that in Chapter 5, and an example is included on the CD ROM.

The 3rd Edition of the book included eVB for Windows CE and discussed actual coding using the ceComm control, various design, development, and debugging issues, along with performance considerations in a WindowsCE hand-held or embedded PC environment. The accompanying eVBTerm and ceVoltmeter examples are used to illustrate this area. This text has been moved to the CD ROM.

Chapter 10 examines the use of direct I/O port manipulation to do things that cannot be done using more conventional API methods. This requires that you understand the physical I/O structure of your PC. Four methods are discussed. The first uses VBASM.DLL (included) to access PC I/O ports on a system using a 16-bit version of Visual Basic. The second uses WIN95IO.DLL to do the same using a 32-bit version of Visual Basic.

Two programs are included that monitor the status of the Carrier Detect bit in the UART. This allows a Visual Basic program to record the total connect time of another Windows program. A third program is used to overcome the maximum speed limitation (19.2k bps) of MSCOMM32.OCX furnished with VB 4.0. Included are examples that are mainly oriented toward Internet applications but also use the Windows and RAS APIs to do some of the actions listed.

Scientific Software Tools, Inc. DriverLINX PortIO software is a freeware DLL and kernel mode driver that permits direct access to standard I/O ports under Windows 9x/Me and Windows NT/2K/XP. It is included on the CD ROM. An example program is provided that allows a program to force DTR false, thus causing a modem to disconnect on a comm port that has been opened by another application. Another example illustrates sending data **directly** to the serial port UART, which bypasses the Windows serial API. While not everyone's "cup of tea," these examples may help solve a problem that otherwise is intractable.

Chapter 11 is important. Debugging communications applications can be difficult. Here are discussed both hardware and software methods for debugging your applications.

Techniques for optimizing your code and tips to increase its reliability are presented. The use and utility of telephone line simulators are discussed. Last, debugging serial port hardware problems are discussed.

Appendix A is a list of resources. Contact information is provided for all of the products that were mentioned in the book. Lots of additional contact information is provided for companies that offer products or services that may be useful but that could not be discussed in detail.

Appendix B is a VT100 terminal emulator written using VB6. Earlier editions of the book included 16-bit VB2 code (still on the CD ROM in the 2ndEdition folder).

Appendix C is the complete NMEA-0183 protocol used in the GPS receiver program.

Appendix D details the I/O port description of 8250 and 16550AF UARTs.

Appendix E is a chart of the ASCII character set.

Appendix F is the basic AT modem command set.

Appendix G is a description of the Telocator Alphanumeric paging protocol.

Visual Basic and Windows Versions

Visual Basic is now in its ninth version and Windows in its umpteenth version. I'll discuss some of the issues with various versions here and in more detail in other sections of the book.

Windows 3.0 was furnished with a device driver (COMM.DRV) that implemented interrupt driven serial communications. This and later drivers are "virtual device drivers." That is, they provide a controlled interface to the services provided to support serial communications and isolate the application from the actual hardware. The Windows 3.0 COMM.DRV did not provide support for 16550 AF UARTs (discussed later) so it was not reliable at speeds in excess of 19200 bps.

Windows 3.1 was furnished with an improved version of COMM.DRV. It provided built-in support for 16550 AF UARTs, an improved notification scheme, and was capable of reliable communications up to 57600 bps.

VB 1.0 had no built-in support for serial communications. If you needed serial communications, you had to rely on the Windows API. This was a fairly complex process, as can be seen from the API chapter in this book. VB would work with Windows 3.x in either Standard or Enhanced mode.

It would be a mistake to do any serial communications under Standard mode Windows. True multitasking is needed to avoid loss of data when a serial communications application runs in the background. So, do not try to write any serious serial communications application that runs under Windows 3.x Standard mode.

There were a few DLLs (Dynamic Link Libraries) that encapsulated the API but these were not too common nor did they work too well.

It is possible to encapsulate the Windows communications API functions in a VBX or OCX and to make those functions easily available to VB programmers. VB 2.0 Professional Edition was furnished with MSCOMM.VBX that provided a simplified communications interface. MSCOMM.VBX also provided event driven communications. Event driven comm is desirable because it means that communications routines can be written in an analogous way to the routines that are associated with other VB events, e.g., CommandButton Click events.

VB 2.0 Standard Edition, and later Standard Editions of Visual Basic, had no built-in support for serial communications. You could use the Windows API functions or purchase a commercial communications add-on.

At the time of the VB 2.0 release, commercial vendors started to offer VBXs that encapsulated the communications API and that offered other enhancements. Some of these enhancements were built-in error-checked file transfers and terminal emulation.

VB 3.0 followed VB 2.0 by only six months. VB 3.0 Professional Edition was also furnished with MSCOMM.VBX. The new MSCOMM.VBX used a feature of the Windows 3.1 API that permitted communications event notification of the control. However, it was soon found (within a few days!) that this event notification did not work reliably at speeds higher than 9600 bps. A new version of MSCOMM was released to deal with this fault. See the section on VB 3.0 and MSCOMM for more details on this problem.

Windows for Workgroups 3.11 was released in this same time frame. WFW 3.11 introduced a new problem. The communications driver (COMM.DRV) that was furnished with WFW 3.11 had a fault of its own. It did not properly identify 16550 AF UARTs. This meant that it would attempt to enable the FIFO (First In First Out buffer) on the UART when, in fact, that FIFO did not exist. This caused unreliable processing of receive data. There were several solutions for this problem. The first involved replacing COMM.DRV with the COMM.DRV from Windows 3.1. The second solution involved editing the SYSTEM.INI file to disable the FIFO for any port that was known to not have a 16550 AF UART. The third possibility was to replace COMM.DRV with a commercial replacement. You can see the Resources appendix for TURBOCOM.DRV from Pacific CommWare. This driver is my preference for solving this problem because it works and because it is a higher-performance driver than the Microsoft-furnished COMM.DRV. Another such driver is HiCom/9 from Cherry Hill Software Corporation.

Windows NT (later Windows 2K and XP) and Windows 95 (later Windows 98 and Me) are 32-bit operating systems. These various operating systems improved serial port handling with each new version (with a few warts here and there). TAPI became viable only under these OS's, and serial port drivers have been updated to improve "Plug and Play" operation with a variety of serial devices. The chapter on Debugging will discuss some of the problems that may be encountered under these OS's.

VB 4.0 introduced several new features and issues. This version of VB came with two development environments. VB 4.0/16 was for Windows 3.x (although, of course, 32-bit versions of Windows would also run VB 4.0/16 programs) and VB 4.0/32 was for Windows 95/98 and Windows NT 3.51 or later. Accompanying the Professional and Enterprise Editions of VB 4.0 was MSCOMM16.OCX for use with VB 4.0/16 and MSCOMM32.OCX for use with VB 4.0/32. From the standpoint of the VB programmer, OCXs (OLE custom controls, where OLE stands for Object Linking and Embedding) were just like the earlier VBXs. These new controls offered no new features but they did cause some new problems that were not seen in earlier versions. See the section on VB 4.0 and MSCOMM for more specific details. To solve some of the problems, new versions of both controls were released on 1/26/96. These updated controls were part of the unadvertised VB 4.0a release of VB 4.0.

VB 4.0 introduced a new issue with respect to binary data and the use of the String data type for Binary data. See the section on VB 4.0 and MSCOMM for an extended discussion on this area of concern. I discuss the best solution for this potential problem also.

VB 5.0, VB 6.0, and VB.NET are 32-bit development environments, only. A new version of MSCOMM32.OCX was furnished with VB 6.0. The VB5 and VB6 versions had some features that were new and very useful. I go into detail on some of these in the VB 5.0/VB 6.0 section. The VB 6.0 version of MSCOMM32.OCX does not offer any features that were not in VB 5.0. So, discussion of VB 6.0 will be combined with that for VB 5.0.

The Visual Studio .NET 2002 and 2003 Frameworks introduce many new functions that are designed to improve programmer productivity. As its name implies, it is largely oriented to development for the still emerging Internet-centric applications. Many of the new features offer value – but not too much that is directly applicable to serial communications. The GPS example application that is included in the VB.NET chapter illustrates the use of a .NET built-in library to perform calculations involving Daylight Savings time. The XMTermNET application illustrates the use of COM interoperability with the XMCommCRC control and the use of ActiveX controls in .NET. A similar VBTerm example, using MSComm32, also is provided on the CD ROM.

The NetTerm example illustrates.NET methods for implementing serial communications without using an ActiveX controls. Instead it uses the .NET FileStream class to read and write serial data using a communications port that has been opened using Platform Invoke (P/Invoke) to call a variety of underlying Windows APIs. The DesktopSerialIO dll that I provide on the CD ROM is an improvement on these techniques.

Visual Studio .NET 2003 Professional and higher Editions include the Compact Framework for PocketPC 2000 and higher devices, and other Windows CE systems using WinCE 4.x and higher. The Compact Framework does not provide any serial communications support, so I provide a complete, practical, serial communications class (DLL). The source code is included, so any modifications that are vital for a specific project may be made. I also furnish example code that illustrates the use of this class.

Visual Studio .NET 2005 includes native serial communications. Explanation and examples are included. See the CD ROM for Compact Framework 2.0 examples for the Pocket PC.

Conventions and Style

There are so many decisions to make when you write a book. One of the most important is how much humor to display? I have found (many, many times) that things that amuse me are not even slightly funny to others. So, be forewarned. If I say something stupid, write it off to my strange world-view. If you give me the benefit of the doubt, I will not have to work so hard.

The word “data” is a plural noun. However, common usage assigns a singular meaning to the word. Grammar dictates that the plural form of a verb should accompany plural subjects in a sentence. The choice made in this book is to follow common usage, rather than the grammatically correct form. So, the word data will be used in sentences with a singular form of the verb.

Microsoft and other knowledgeable authorities suggest that Hungarian notation should be used for variable and objects. For example, a TextBox might be named txtReceiveData. A string variable might be named sBuffer, and an integer variable might be named nTotalCount (or iTotalCount, depending on whom you ask).

The rationale behind these naming conventions is that one can tell from the name what the variable type is or what kind of object is named. This can simplify debugging of your own code, simplify maintenance, and make the code more readable by a casual viewer. All of these arguments, and perhaps others, are valid.

However, I often do not use Hungarian notation. I try to use descriptive names for my variables and objects. Habits of twenty years or so are hard to break and the extra prefixed characters that describe type or function are not natural to me. I apologize in advance for this failure and hope that it will not be too great a limitation. I do not use the default property of controls for an assignment. If one were to do so, readability would be enhanced by using Hungarian notation. However, statements like ReceiveData.SelText = Buffer is rather unambiguous. I do use Hungarian-like notation for private variables inside class modules and at various other times – I have tended toward verbosity as the years have gone on. This is my own idiosyncrasy. It reminds me of their use and allows me to use similarly named property Let and Get names.

Occasionally, I use a variable type suffix. For example, I might use Ret% to designate an integer or Buffer$ to designate a string. When writing code, I do this on an ad hoc basis (and have abandoned it in recent years). I suggest that you decide on your own coding style or use the standard that your organization has adopted. However, these suffixes are not supported in VB.NET. For that reason, most of the code that has been modified for the 3rd and later editions of this book removes these suffixes.

Some of the code examples in this book use Chr$(13), the Carriage Return character, instead of the equivalent VB intrinsic constant vbCr. The reason is that versions of Visual Basic earlier than 5.0 did not provide this and other built-in intrinsic constants. I like to use vbCr and vbCrLf when possible. However, you may see either form in the book text. VB.NET changes the syntax for these constants, so code written for .NET may use syntax like ControlChars.Cr for the Carriage Return character. VB .NET still supports the VB Constants vbCr and others, so you can choose the syntax that you prefer. See the .NET Help system for more information.

I always enable Option Explicit (Require Variable Declaration) and I suggest that you do so, too. I explicitly type variables. If a variable must be converted from one type to another, I use an explicit conversion and do not use any form of variant conversion.

Variants and conversion of types to variants can be troublesome. Variant variables require more memory, are slower to access, and their use can cause unpredictable errors. VB.NET no longer supports Variant data types. This reinforces the idea that Variants should not be used. VB.NET adds a compiler directive called "Option Strict." Option Strict restricts implicit data type conversions to only *widening* conversions. This explicitly disallows any data type conversions in which data loss would occur and any conversion between numeric types and strings. I suggest that you employ Option Strict On. It means that your code might be more verbose. However, the added safety is worth the effort.

I use Public or global variables only when they make sense. Artificial data hiding that requires extra parameter passing can make code maintenance easier but, contrary to some popular opinion, it does not make code more "modular." However, the use of global variables can cause side effects. So it is best to limit their use as much as is practical.

The sample code that is furnished with this book is as compact and concise as I could make it, within reason. I have kept the user interface as simple as possible. I have limited the number of forms any project uses to the minimum number that would work while providing the functionality that I desire.

One of the decisions that I had to make was what to call the sample apps that I furnished. I decided to call them applets to indicate that they are complete but that their functionality is limited. I hope this decision will not cause them to be confused with the applets that Microsoft furnishes with various operating systems and programming environments.

In some of the simpler programs, I hard-code some variables that a production application design would allow the user to configure. For example, the NIST Automated Time Program and RingDetect programs are hard-coded to Com1. Both of these programs run minimized, without normal user interface.

A production application will most often have a user interface that will allow the user to configure all critical variables. Such a production application will persist configuration variables in INI files, the Registry, or in a database. These configuration parameters will be read and used when the program is run subsequently. Several of the programs in this book illustrate that technique while others do not have this feature.

When I discuss step-by-step approaches to problem solving, usually I will do one of two things. I may present "pseudo code." This is a code-like expression of the steps that might be used. It is not real code. The other approach that I will use is to present code, based on MSCOMM, which could be compiled. Pseudo code allows me to present a concept without getting into the issue of making certain that the syntax is exactly correct. While real code is concrete, it takes more work to make certain that it will actually work as described. If I present code based on MSCOMM, it should work unchanged with all commercial communications custom controls. However, if you use a DLL-based product, you may have to make some changes in syntax but the logic will often be the same.

When I had a choice about placing subroutines or functions in a form or in a .BAS module, and if I did not need a .BAS module for other reasons, I placed the code in the form. The reason for this decision is twofold. First, fewer files are required to make a functional program, thus distribution of the source code is easier. Second, it is easier to view code on screen and on the printed page when associated procedures are close together.

A good programming technique is to use numeric constants with easy-to-understand names. Visual Basic includes CONSTANT.TXT with a number of these pre-defined constants. For example, an OnComm receive data event can be tested for using Const MSCOMM_EV_RECEIVE = 2. If MSCOMM1.CommEvent = MSCOMM_EV_RECEIVE Then (do something). By all means, use constants to make your code easier to read and maintain.

When later versions of Visual Basic were introduced, the name of some constants changed but not their meaning or value. For example, the MSComm CommEvent constant that indicates receipt of data in an OnComm event is comEvReceive for MSComm32.OCX (version 5 and later) while it was called MSCOMM_EV_RECEIVE in earlier versions. The actual value, 2, was retained. You may see either variation of these constants in the example code furnished with the book.

Public Enums extend the concept of providing easily interpreted constants, such as those used by MSCOMM and all of the classes in the .NET Framework.

Simply said, some of my code examples have no class. By that I mean that many of my examples do not use class modules. Classes enhance code reusability and maintenance. However, the reason that they enhance reusability and maintenance is that they can obscure details of the underlying design and implementation. These details are exactly what I want to emphasize in this book. However, I include a number of examples that do use class modules. One notable example is the source code for the ActiveX custom control XMComm. ActiveX projects must use class modules for their public interfaces so they are a natural to illustrate modular, object-oriented design. Visual Studio .NET is truly object-oriented. Use class modules when programming for .NET. This is a natural and inevitable element of design/implementation. When you create your own applications, I encourage you to use classes where appropriate. The Resources appendix lists a couple of books that discuss this in detail.

I should make one comment on the typographical conventions that I have adopted. I have decided to paste code from my Visual Basic projects into the examples sections of the book. Some of the sample code "wraps" from one line to line to the next. If it is typed in exactly as it appears, it will not run in 16-bit versions of VB. I could format it with artificial underscore characters; the line-continuation character that is used in VB 4.0 and later versions, but that is not available in earlier versions. The editing decision has been made to add underscores to indicate a line-continuation in code but not in comments, regardless of the version of VB in use.

On the CD ROM that accompanies the 2nd, 3rd, and 4th Editions of this book, I have included a number of example programs and other files that are not described in the text. This was done in order to keep the book as affordable as possible. I have included a text file that accompanies each of the extra examples that gives a short explanation of their use and utility. I hope that you find them to be useful. Several of these have been furnished by readers and are included with their approval. Here are some of the topics that are included in this supplementary set of files:

- **ShareCom**. An ActiveX EXE (OLE Server) that allows a single serial port to be shared between several Client applications.

- **RASConnect**. A simple VB/32 project similar to CDMonitor in Chapter 8. The difference is that it uses API methods and is limited to DUN or RAS modem connections.

- **RASHangup**. VB/32 code similar to the Hangup programs in the book. This code will only work with RAS connections. However, it does not require an add-on DLL.

- **GPS4VB3, GPS4VB4, and GPS4VB5-6**. GPS projects that further illustrate decoding NMEA-0183 sentences. Included are displays of Universal Time, Latitude, Longitude, Speed, and Course over the ground, Altitude, and Satellites in use.

- **CommProtocol**. An example of a simple communications protocol that might be appropriate for a dedicated application. This sort of protocol might be appropriate for controlling a device or devices on a dedicated serial network where you control the software implementation for all devices. For example, this might be an RS-485 network that communicates with embedded controllers. If you do not have to adhere to any specific standard protocol, this example shows how to implement one that has several desirable features.

- **Comm Spy**. A project written by Leon Kenison, a student in the Computer Engineering Technology programs at New Hampshire Technical Institute. See the accompanying files for more information

- **TapiClass**. A set of class modules, and an example VB5 project that substitutes class objects for Crescent's PDQTapi. TapiClass was written by Will Fookes. These class modules improve on certain characteristics of PDQTapi. These class modules are designed to be used with PDQComm.

- **XMODEM.TXT**. This file describes the XMODEM and Ymodem error-checked file transfer protocols.

- **9BitData**. A short discussion of how to tackle the 9-bit data problem (PC serial port hardware supports data bits up to 8).

- **Access**. Several Access databases that employ the NETComm.ocx to add serial communications.

- **AppendBinaryArrays**. A project that illustrates how to append binary data in arrays, in a way similar to the way one string might be appended to another.

- **ASP**. Illustrates the use of the NETComm.ocx serial communications ActiveX control in a simple web client (browser) application.

- **CreatePacketSend**. A simple communications protocol for a specific serial hardware system (laser table positioner).

- **Excel.** Use of NETComm.ocx to add serial communications to Excel worksheets.

- **FileCompression.** Various file compression methods, including ZIP.

- **FindTAPIModem.** Use TAPI to identify connected modems.

- **IntelToMotorols.** Convert Intel floating-point values to Motorola floating-point.

- **Scales.** VB and VBA examples that interface with various scales (industrial weighing devices).

- **WindowsMobile5.** Compact Framework code for Visual Studio 2005.

- **GSM INTERFACE SPECIFICATION**. Cellnet's Short Message Service Centre (SMSC) TAP interface provides external companies with the facility to submit short messages of up to 160 characters to GSM mobile subscribers and subsequently to determine the status of those messages. I have added a folder specifically for the 3rd and later Editions that have additional files, folders and utilities. Here are some of these:

- **Zmodem.txt**. This file contains a description of the Zmodem file transfer protocol. It could be used to help a reader implement this protocol. However, anyone really interested in this should consider the comments that I make on this subject in the chapter that deals with file transfer protocols

- **Breakout** is a complete setup program for a utility that I have written. It uses direct port I/O to read the serial port UART modem control and modem status registers. The resultant data is displayed both as a state indication (similar to the LEDs on a breakout box) and as a time-domain logic scope display of the status of the RTS, DTR, CTS, DSR, CXD, and RI lines of the serial port. The Breakout program works with all 32-bit versions of Windows (9x/Me and Windows NT/2K/XP). See the README file that is included for a complete description.

- **SDA322** contains source code for two projects that work with the B&B Electronics (see Appendix A) SDA line of serial data acquisition modules. One project is an ActiveX control that provides an easy way to interface to these modules for PC-based applications. The other is an ActiveX EXE that provides a similar interface that might be called from a Visual Interdev (ASP) or equivalent application so that the same SDA data can be accessed by a browser via the Internet.

- **mComm** contains a class module that encapsulates MSComm32.ocx. The unusual aspect to this code is that it allows MSComm to be used **without** the need to place an instance of MSComm on a form. I have had lots of people ask for this (though, I admit, I have never found a need to do this).

- **mCommTLB** is a class module that encapsulates MSComm32.ocx (see above), but that uses a Type Library implementation of the ClassBuilder to eliminate licensing issues seen when using MSComm without a form.

- **CalculateEvenParity** illustrates parity calculation in code.

- **ConversionRoutines** illustrates numeric conversions that may be needed when dealing with external systems.

- **EnumPorts** enumerates installed hardware ports. These include serial, parallel, and network ports.

- **SendMail** is an illustration of various network email functions. This code was written by Monte Hansen.

- **SimpleMAPI** is an illustration of MAPI written by Michael Kaplan.

- **X10** contains information and examples covering the popular X10 home networking/control system.

- **NETComm** is the source code form an ActiveX control that wraps the functionality of MSComm32.ocx, so that it may be used with on license restrictions in Visual Studio .NET, Access, Excel, Visio, or other ActiveX clients. Simple Access database and Excel examples are included. NETTerm is a terminal example that illustrates using NETComm.ocx in VB .NET.

- **VT100** is a VB6 terminal emulation that employs MSComm32.ocx.

- **VBTerm.net** is a port of the VBTerm example to .NET. This port used the Upgrade Wizard, though I made a few manual changes.

- **DesktopSerialIO** provides a simple but powerful serial object for Visual Studio 2002/2003 (located in the Chapter 5 folder).

- **XMCommNET** is the XMODEM file transfer protocol implemented using calls to the Windows serial API and is based on the serial code in DesktopSerialIO (located in the Chapter 5 folder).

- **VirtualSerialPort and DataMonitor.** This is a utility that I wrote to use a pair of virtual serial ports **and** a hardware serial port to facilitate testing and debugging of serial applications. Find it in the Serial Communications Products folder under HardAndSoftware on the CD ROM.

If there is a feature that you want to use in your application that is not implemented in one of the programs in this book, it may be in another. So, look around.

Acknowledgments

I have had a number of people who have added considerably to the technical content of this book. I would like to thank them all, in no particular order.

Daniel Appleman provided the original code for the 32-bit Windows API chapter. The API32Term program appeared first in his book *Visual Basic Programmer's Guide to the Win32 API*, ZD Press. I highly recommend his book for anyone who needs to program using the API. This book follows the popular 16-bit version *Visual Basic Programmer's Guide to the Windows API*, ZD Press. I took the liberty to modify the program to match my preferences and prejudices but I appreciate the knowledge that Dan has shared with the VB community.

To compile a practical application that incorporates file transfers or emulation, you may want to purchase a commercial communications add-on. A number of such products are described in the book. Any of these might be used, with appropriate modification of the code that I have furnished, which rely on Sax Comm Objects. Often these modifications will be minor and I will outline the differences between add-ons in each section.

Ron Tanner furnished the 16 and 32-bit Power Page DLLs that are included on the CD ROM for alphanumeric paging. However, these DLLs are not supported.

Jonathan Wood furnished the VBASM.DLL for 16-bit applications. This DLL complements the WIN95IO.DLL that I have included on the CD ROM for 32-bit access to I/O ports under Windows 95/98. VBASM.DLL is freeware. Thanks to SoftCircuits and Jonathan Wood.

Jim Stewart of JK Microsystems furnished a Flashlite Single Board Computer embedded PC so that I could develop the Flashlite Control program. The quid pro quo was that JK Microsystems could include the program that I developed for the Flashlite on their Utilities disk and on their Web site, to benefit all of their customers. Fair all around, I think.

I want to thank Jim Mack of MicroDexterity for the BuffToHex functions in the RGUTIL16.DLL and RGUTIL32.DLL files that I have included on the CD ROM (see the Debugging chapter). MicroDexterity sells the outstanding product named Stamina. It features many add-on functions that can be invaluable for serial communications programs.

Karl Peterson, a friend and fellow MVP, has provided dozens of code examples and ideas on his web site (www.mvps.org/vb) that are valuable. I have incorporated one or two of his ideas in the code in this book.

I want to repeat my appreciation to James Shields and Ruth James at Mabry Software. Without their support I would not have been able to publish this book.

Last, I thank all readers of the First, Second, and Third Editions of this book who took the time to contact me with improvements and corrections. I would like to give special thanks to Ronald Frakes, Terrance Simkin, Will Fookes, and Jeffrey Schnell. I received valuable comments on the Second Edition from Dan Karmann and Fabio Varriale. Valuable input from John Kozee was used in the Fourth Edition. Thank you.

How to Contact Me

Richard Grier
Hard & Software
12962 West Louisiana Avenue
Lakewood, CO 80228

303-986-2179 (voice)
(303)593-9315(fax or voice mail)
Dick_Grier@msn.com (email)
dick_grier@hotmail.com (email)
www.hardandsoftware.com
www.hardandsoftware.net