# PDQComm™ for Windows

Technical Support  Copyright and Trademarks  Distribution

Welcome and thank you for purchasing PDQComm for Windows from the Crescent Division of Progress Software Corporation. PDQComm for Windows is the premier tool for developing communcations applications using Microsoft Visual Basic.

PDQComm for Windows: An Overview


PDQComm Custom Control
Using the PDQComm Control
PDQComm QuickStart
PDQComm Control Reference

ModemWare
Using ModemWare
ModemWare Tutorial
ModemWare Functions, Subroutines, and Variables -- A to Z


ModemWare Scripts
ModemWare Script Commands -- A to Z


Hayes Modem Commands

# ModemWare Functions, Subroutines, and Variables - A to Z

A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

Related Topics

**A**

AsciiSend

**D**

Download

**H**

Hangup

**L**

LoadPortSettings
LoadTerminalSettings
LoadModemSettings

**M**

MW_CancelFlag
MW_CurRoutine
MW_DisplayFlag
MWEnd
MWError
MWErrorMsg
MWPause
MWPhoneBook
MWSetError
MWSetProtocol
MWStatus

**O**

OpenPort

**P**

PlaceCall

**S**

ScriptPlay
ScriptRecord
SendModemCmd
SetPortSettings
SetTerminalSettings
SetModemSettings

**U**

Upload

**W**

WaitFor
WaitForA
WaitForCall


# ModemWare Script Commands

Related Topics

**Symbols**

:LABEL Command

**C**

CAPTURE Command
CLOSECAPTURE Command

**D**

DIAL Command
DOWNLOAD Command

**E**

END Command

**H**

HANGUP Command

**I**

INPUT Command

**O**

ON_TIMEOUT GOTO Command

**P**

PAUSE Command
PORT Command
PROTOCOL Command

**S**

SEND Command
SETTINGS Command
STOP Command

**T**

TIMEOUT Command

**U**

UPLOAD Command

**W**

WAITFOR Command

# Getting Technical Support

The Crescent technical support staff is ready to help you with problems that you encounter when installing or using PDQComm. It does not matter what component of PDQComm you are having a problem with; the Crescent technical support staff will do its best to help you succeed with PDQComm.

If you need technical support, contact Crescent using any of the following methods:

| | |
|---|---|
| **By Telephone** | Contact Crescent's North American technical support staff at (**617) 280-3000** -- Monday through Friday from 9:00 a.m. to 5:00 p.m. EST. |
| **By FAX** | Contact Crescent by FAX at (**617) 280-4025**. |
| **Via BBS** | Contact Crescent through our 24-hour bulletin board service at (**617) 280-4221**. |
| **Via CompuServe** | Contact Crescent through CompuServe address **70662,2605**<br><br>Crescent also maintains a section in the MS Windows Components A+ Forum on CompuServe. To reach the Crescent section, type the following at the CompuServe prompt:<br>**GO CRESCENT** |
| **By Electronic Mail** | Contact Crescent using the Internet address **crescent-support@progress.com** |
| **Via the WWW** | View the Crescent Web page at **http://www.progress.com/crescent** |
| **By Mail** | Address your correspondence to:<br>**Technical Support**<br>**Crescent Division, Progress Software Corporation**<br>**14 Oak Park**<br>**Bedford, Massachusetts 01730** |

Please have your product name, version number, serial number, and system configuration information available so that the Crescent technical support staff can process your support requests as efficiently as possible.

# PDQComm for Windows: An Overview

Technical Support  Copyright and Trademarks  Distribution

PDQComm for Windows is the complete package for building 16-bit and 32-bit serial communications applications with Microsoft Visual Basic 4.0. The two main components of the PDQComm product are:

- The PDQComm control
- ModemWare

The following topics summarize the features of the PDQComm for Windows product:

- General Features
- PDQComm Control Features
- PDQComm and MSComm: A Comparison
- ModemWare Features

The following topics provide additional information about the PDQComm for Windows product:

- Hardware and Software Requirements
- PDQComm Product Files
- File Distribution and Application Deployment
- PDQComm Demonstration Programs

# The PDQComm Control

Related Topics

The heart of the PDQComm for Windows product is the PDQComm control. This custom control embeds serial communications functionality into a Visual Basic application, allowing the application to transmit and receive data through a serial port.

The PDQComm control is an enhanced version of the MSComm control that comes with the Professional Edition of Microsoft Visual Basic 4.0. In addition to all the features of MSComm, the PDQComm control has built in terminal emulation, support for the popular file transfer protocols, and a number of other useful features. Because we developed the MSComm control for the Visual Basic product, PDQComm is compatible MSComm.

PDQComm uses the Windows API and communication functions to handle serial communications. This means that PDQComm supports any serial port hardware that works with Windows. Most extended serial port card manufacturers supply a Windows driver for their hardware, which makes the device available to all Windows applications that use the Windows API for serial communications.

# ModemWare

Related Topics

ModemWare is a suite of high-level Visual Basic routines that greatly simplifies the development of applications that control modems. These routines and dialogs allow you to initialize modems, place calls, disconnect, wait to receive strings from a serial port, and more.

With ModemWare, you can easily write applications for portable computers with cellular phones. You can access data service networks such as Compuserve, MCI, or Dow Jones. Your user never has to see a terminal window, simply call the service, do your thing, and hang up.

Serial port communications programming was revolutionized with the advent of MSComm and PDQComm. Still the headache of dealing with modems requires even experienced Visual Basic programmers to write a lot of support code. Not any more! ModemWare makes communications programming as close as you can get to plug and play modem support.

# PDQComm for Windows: General Features

Related Topics

- Access any communications port that is available to Windows. If TERMINAL.EXE can access the port, so can PDQComm for Windows.


Multi-port access allows you to use more than one port at a time.


Terminal emulations included are TTY, ANSI, VT-100, and VT-52.


File transfer protocols included are XModem-Checksum, XModem-CRC, XModem-1K, YModem-G, YModem-Batch, ZModem, Compuserve B+ and Kermit.


File transfers occur in the background.


A built-in file transfer dialog box displays vital information, such as bytes transmitted, time elapses, and file size.


XON/XOFF and RTS/CTS handshaking are fully automatic.


Transmit and receive buffer sizes are adjustable.

# PDQComm Control Features

Related Topics

A superset of the functionality provided by the MSComm control contained in the Professional Edition of Microsofts Visual Basic 4.0.

OCX support for VB4 (16- and 32-bit).

A complete terminal program can be written without writing a line of code. Just set a few properties, and it works!

Automatic ZModem allows ZModem file transfers to start automatically.

An adjustable scroll-back buffer for viewing received data.

Clipboard access for copying and pasting text from the terminal window.

Allows trapping of all communications events (such as received data and change in carrier detect, etc.) and errors (such as receive buffer overflow, overrun, or CTS timeout).

# PDQComm and MSComm: A Comparison

Related Topics

The MSComm control has a subset of the functionality in the PDQComm control. PDQComm has the following enhancements:

**Simple Communications**

PDQComm supports baud rates up to 230,400 with the appropriate Windows communication driver.

**File Transfers**

PDQComm supports the following; XModem-Checksum, XModem-CRC, XModem-1K, YModem-Batch, YModem-G, ZModem, Kermit, and Compuserve-B+ transfers.

All transfers happen in the background.

The received file is opened in shared mode so the data can be accessed as the file is being received.

Use the status dialog box to display bytes transmitted, percent completed, time left, etc., or use your own custom dialog box, or none at all.

**Terminal Emulations**

The PDQComm control turns into a text window when you set the Emulation property to supported emulation.

TTY, ANSI, VT-52, and VT-100 emulations built into the PDQComm control.

Uses the standard Windows font control properties.

Contains color properties and settings for monochrome and gray-scale colors.

Employs the standard Windows cut-and-paste method, as well as the SelStart, SelLength and SelEnd properties.

Contains a variable-length scroll-back buffer to allow the user to scroll back through the received data.

# ModemWare Features

Related Topics

Dialog boxes for configuring a serial port, terminal settings, and modem settings. Each of these dialogs writes their settings to a configuration file associated with your application.

A database of modem configuration information with more than 440 entries for commercially available modems.

A powerful script language that allows non-programmers to record and edit scripts that accomplish such tasks as calling a host system, reading mail, and transferring files.

A phone book which manages a list of entries and allows you to dial them.

# Hardware and Software Requirements

PDQComm has the following hardware and software requirements:

- Any IBM-compatible machine with an 80386 processor or higher
- A 3 1/2 floppy drive
- MS-DOS 5.0 or higher
- 2 MB of free space on your hard disk

The 16-bit version of PDQComm has the following hardware and software requirements:

- Microsoft Windows 3.1 or higher, Microsoft Windows for Workgroups 3.11 or higher, Microsoft Windows NT 3.5 or higher, Microsoft Windows 95
- Microsoft Visual Basic 4.0 (16-bit) for Windows and the hardware and software that it requires to function

The 32-bit version of PDQComm has the following hardware and software requirements:

- Microsoft Windows NT 3.5 or higher, Microsoft Windows 95
- Microsoft Visual Basic 4.0 (32-bit) for Windows and the hardware and software that it requires to function

# PDQComm Product Files

The PDQComm installation program installs most product files into the specified PDQComm product directory. This topic categorizes and describes the more important product files that you should know about.

# PDQComm Control Files and Supporting Libraries

Related Topics

The following files define the PDQComm control and permit the use of the control to develop communications applications.

| Files | Description |
|---|---|
| PDQCOM32.OCX | A 32-bit, OLE-compliant, PDQComm control file for use with the 32-bit version of Visual Basic 4.0. |
| PDQCOM16.OCX | A 16-bit, OLE-compliant, PDQComm control file for use with the 16-bit version of Visual Basic 4.0. |
| PDQCOMM.LIC | A license file that allows you to design applications that access and use the PDQCOM32.OCX control file. |

# ModemWare Form Files, Module Files, and Database Files

Related Topics

The following files contain Visual Basic source code for ModemWare routines and data used by these routines.

| File(s) | Description |
|---|---|
| COMDEBUG.BAS | Routines that help you debug applications developed with PDQComm. See the README.WRI in the PDQComm product directory for more information about COMDEBUG.BAS. |
| MODEMS.DAT | The ModemWare modem database file. This file contains settings for over 400 types of modems and supports several of the ModemWare modem configuration routines. |
| MODMWARE.BAS | Core ModemWare functions and subroutines. This file also defines ModemWare variables and structures. |
| MWCAPT.BAS | ModemWare capture routines. |
| MWMODEM.BAS, MWMODEM.FRM | ModemWare modem control routines and the modem settings dialog box. |
| MWNOSCPT.BAS | Stub file used to disable ModemWare scrip functionality. |
| MWPHBOOK.BAS, MWPHBOOK.FRM, MWPHEDIT.FRM | ModemWare phone directory routines and the phone directory dialog boxes. |
| MWPORT.BAS, MWPORT.FRM | ModemWare port control routines and the port settings dialog box. |
| MWSCRIPT.BAS | ModemWare script routines and commands. |
| MWTERM.BAS, MWTERM.FRM | ModemWare terminal control routines and the terminal settings dialog box. |
| MWXFER.BAS, MWXFER.FRM | ModemWare file transfer routines and the file transfer protocol dialog box. |

The PDQComm product directory also contains binary versions of the form files (.FRX) listed in the previous table.

# PDQComm Help Files

PDQComm Product Files

The following help files are available for the PDQComm product.

| File | Description |
|---|---|
| PDQCOMM.HLP | Contains information about the entire PDQComm product, including reference information for the PDQComm control, ModemWare routines, and ModemWare script commands. ***This help file.*** |
| MWSCRIPT.HLP | Provides information about the ModemWare script commands. You can distribute this file with your application as documentation for the script commands. |

# PDQComm Online Documents

PDQComm Product Files

The following documents reside in the PDQComm product directory, supplying information about the current release of the product and supplemental information for some of the communications topics presented in the *PDQComm User's Guide* and this help file.

| File | Description |
|---|---|
| FXFERBAS.WRI | Contains some basic, background information about file transfer operations and protocols. |
| HISTORY.WRI | Provides a history of enhancements and bug fixes across the previous versions of the PDQComm for Windows product. |
| MODEMBAS.WRI | Contains introductory information about modems. |
| README.WRI | Provides information about the current release of the PDQComm for Windows product. |
| TECHHELP.WRI | Contains technical tips and information about how to get technical support for the PDQComm for Windows product. |

# PDQComm Demonstration Files

Related Topics

The PDQComm product directory contains executables and source code files for four Visual Basic demonstration projects. The following table lists the files that are specific to the PDQComm demonstration programs:

| **Demonstration** | **File(s)** |
|---|---|
| PDQComm Control Demonstration Program | CTRLDEMO.EXE, CTRLDEMO.VBP, CTRLDEMO.BAS, CTRLDEMO.FRM/.FRX, ABOUT.FRM/.FRX, BTNDEF.FRM, TOPTION.FRM/.FRX |
| Host Dial-in Program | HOST.EXE, HOST.VBP, MINIHOST.BAS, MINIHOST.FRM/.FRX |
| ModemWare Routine Demonstration Program | MWDEMO.EXE, MWDEMO.VBP, MWDEMO.BAS, MWDEMO.FRM/.FRX |
| Remote Dial-in Program | REMOTE.EXE, REMOTE.VBP, REMOTE.BAS, REMOTE.FRM/.FRX |

These demonstration projects also use ModemWare module files, form files, and the PDQComm control.

# File Distribution and Application Deployment

PDQComm Product Files

You can distribute any application you create using PDQComm routines, as long as you distribute the application only as an executable. You may also distribute the following PDQComm product files with your applications:

| | |
|---|---|
| MODEMS.DAT | PDQCOM16.OCX |
| MWSCRIPT.HLP | PDQCOM32.OCX |

If you are distributing an application built with the PDQCOM32.OCX control, you must also distribute the following support files: OLEPRO32.DLL, MFC40.DLL, and MSVCRT40.DLL.You cannot distribute any PDQComm source code or routines in such a way that other people can reuse them. Further, you cannot distribute the PDQCOMM.LIC file.

See the PDQComm End-user License Agreement at the beginning of the *PDQComm User's Guide* for more information about the license to use PDQComm for Windows.

# PDQComm Demonstration Programs

PDQComm Demonstration Files

The PDQComm installation program installs four demonstrations programs into the PDQComm program group. The following table provides information about the PDQComm demonstration programs:

| Demo Icon | Project File | Description |
|---|---|---|
|  | CTRLDEMO.VBP | A complete terminal program written using ModemWare routines. This demonstration shows you how to handle file transfers and other OnComm events. This program uses every aspect of PDQComm for Windows. You can record and play scripts, use the phone, and much more. |
|  | HOST.VBP | A host (dial-in) program. |
|  | MWDEMO.VBP | A demonstration that contains the proper syntax for calling the ModemWare routines and testing for errors in a useable context. |
|  | REMOTE.VBP | A remote (dial-in) program. |

The executables and source files for these demonstration program reside in the PDQComm product directory. These demonstrations illustrate many of the concepts and techniques described in this help file. They also serve a library of code that you can use to build your own communications applications.

PDQComm Product Files
PDQComm Demonstration Programs

# Using the PDQComm Control

Related Topics

The PDQComm custom control is the most sophisticated programming tool for developing serial communications applications on the market today. The CTRLDEMO.VBP project, located in the PDQComm product directory, provides a full demonstration of the power and features of the PDQComm control.

This topic provides access to information about communications concepts, programming tips, and some common tasks that you can use the PDQComm control to perform in a communications application.

# PDQComm Control: Quick Start

Related Topics

The instructions in this topic show you how to create a simple terminal program with only one line of code using the PDQComm control.

**NOTE** These instructions require the following software and hardware: Visual Basic 4.0, PDQComm 3.0, and a modem.

---

**To create a simple terminal program, do the following:**

1. Start Visual Basic 4.0.
2. From the Tools menu in Visual Basic, select the Custom Controls option to load the PDQComm control. Load PDQCOM16.OCX if you are running the 16-bit version of Visual Basic or load PDQCOM32.OCX if you are running the 32-bit version of Visual Basic.
3. Add a PDQComm control to Form1 by double-clicking on the PDQComm icon on the Visual Basic Toolbox.
4. Set the CommPort property to a valid communications port number. Use the Ports utility in the Windows Control Panel application to see the available port numbers.
5. Set the Settings property to 2400,N,8,1 for a 2400-baud (or greater) modem, or 1200,N,8,1 for a 1200-baud modem.
6. Set the AutoProcess property to 3 (both serial and keyboard input).
7. Set the Emulation property to 2 (ANSI).
8. Set the AutoSize property to True.
9. Add the following line to the Form_Load event of Form1:

   ```
   Comm1.PortOpen = True
   ```

10. From the Tools menu in Visual Basic, select the Options menu option and make Form1 the startup form for the current project.
11. Run the project.

When you run the project, it takes control over the modem or serial device associated with the specified communications port. If the port is associated with a modem, you can type modem commands that tell the modem what to do. For example, the following command dials the Crescent BBS:

```
ATDT 16172804221
```

See the topic Hayes "AT" Modem Commands for more information about modem commands.

If a Device Unavailable error occurs when you run the project, the port specified by the CommPort property is not available to Windows. Try experimenting with the settings 1 through 4 for the CommPort property. If there are no available ports, you then need to refer to your *Microsoft Windows Users Guide* for information on how to set up communications ports in Windows.

# Control Properties, Methods, Events, and Constants

Related Topics

Just like any other custom control for the Visual Basic environment, the PDQComm control has properties, methods, and events. These elements of the PDQComm control give you complete control over all aspects of communications. If you are not familiar with using custom controls in Visual Basic, consult the *Microsoft Visual Basic Programmers Guide*.

The PDQComm control uses constants to represent result values and error condition values. Also, several properties of the PDQComm control take enumerated values. These properties can only accept values from a discrete list of integer values. Each integer value represents a behavior setting for the property. To make your code more readable, you can use predefined constants for the PDQComm control to set and analyze the value of an enumerated property. The .OCX version of PDQComm, automatically defines a set of constants to represent results, errors, and enumerated property values.

See the PDQComm Control Reference for complete documentation of the constants, properties, methods, and events of the PDQComm control.

# Receiving Data and Using The OnComm Event

Related Topics

The OnComm custom event provides a way to trap all communications errors and events. PDQComm assigns constant values to all possible Windows communications errors and events. When the OnComm event occurs, PDQComm sets the CommEvent property to the constant representing the communications error or event that triggered the OnComm event. For example, if the OnComm event occurs and the CommEvent property equals PDQ_EV_RECEIVE (received data), PDQComm received some characters. See the CommEvent property for a list of communications errors and events.

To trigger an OnComm event when PDQComm receives characters, you must set the RThreshold property to the number of characters to be received before the OnComm event is fires. For example, if you want the OnComm event to occur each time PDQComm receives a character, set the RThreshold property to 1. Set the RThreshold property to 10 to trigger OnComm event after PDQComm receives every 10 characters. See the RThreshold property for more information.

There are two other methods for receiving data:

You can poll for data manually using the InBufferCount and Input properties.

```
This demo displays all input in the terminal window.

Comm1.InputLen = 0
Do
   Dummy = DoEvents()
   If Comm1.InBufferCount Then
      In$ = Comm1.Input
      Comm1.Disp = In$
   End If
Loop
```

You can tell PDQComm to automatically display the data it receives by setting the AutoProcess property to PDQ_AUTOPROCESS_SERIAL (serial input) or PDQ_AUTOPROCESS_BOTH (both keyboard and serial input):

```
This demo displays all serial input in the terminal window.
Comm1.AutoProcess = PDQ_AUTOPROCESS_SERIAL
```

# Binary vs. Non-Binary Mode

Related Topics

There are three common options for opening a communications port:

**Binary Mode (BIN)**

All data is transmitted verbatim. Tab characters do not expand and Chr$(26) is not recognized as an EOF (end of file) marker.

**ASCII Mode (ASC)**

Tab characters expand and Chr$(26) is recognized as an EOF marker. A carriage return (Chr$(13)) is automatically inserted after every 80 characters.

**Linefeed Mode (LF)**

This option is normally used in with ASCII mode. It inserts a linefeed (Chr$(10)) after every carriage return (Chr$(13)).

PDQComm for Windows only supports binary mode, due the limited usefulness of the ASCII and linefeed modes.

# Multiple Port Operation

Related Topics

Accessing multiple ports from a single application is easy with PDQComm. Just use a different PDQComm control for every port you want to access. Each control manages a single serial port. The number of serial ports that you can safely access simultaneously is highly dependent on the speed of your systems CPU and the number of processes running under Windows.

To access more than two ports simultaneously, you need to purchase a 16-bit serial port card that comes with a driver that works with Windows (and by extension with PDQComm for Windows).

Two major OEMs that provide such hardware and software are:

| | |
|---|---|
| DigiBoard | 800-344-4273<br>612-922-4287 |
| Comtrol | 800-926-6876<br>612-631-7654 |

# Handshaking and the Receive Buffer

Related Topics

When serial data is exchanged, characters are sent in a continuous stream from one device to another. Each character received must be read immediately or it will be lost when the next character arrives. A special area of memory, called the receive buffer, is used to hold characters between the time they are received and the time they are read. This temporary storage allows time to pass before a character is read and frees the software to perform other tasks while data is being received.

The job of a communications library, such as PDQComm for Windows, is to create the receive buffer and to place characters in it for a program to later read. If a program reads data from the receive buffer slower than the data is being placed into the buffer, the buffer will become full and any further data received will be lost. Therefore, another arrangement is often used to signal the sending device to stop when the receivers buffer becomes full. The sender is also signaled to resume when the receiver is again ready. This arrangement is called handshaking or flow control.

PDQComm for Windows supports hardware (RTS/CTS) handshaking, as well as software (XON/XOFF) handshaking. Hardware handshaking uses signals conducted through wires, while software handshaking uses the exchange of ASCII control characters (CTRL-S means stop sending, and CTRL-Q means resume). Hardware handshaking can be used only when the two devices are connected by RS-232 cables or when using high speed modems; however, it is able to control the transfer of any type of data. Software handshaking does not require the devices to be directly connected, but it cannot control the transfer of binary data because the difference between control characters and binary data is unrecognizable.

Both the sending and receiving devices must use the same method (hardware/software) of handshaking. The method of handshaking is specified when PDQComm for Windows opens a communications port. Once that is done, no further effort is required by the receiver to perform handshaking. Signals will automatically be sent by the receiver depending on the state of its receive buffer. However, a small amount of extra effort is required by the sender, to be sure that data is properly sent when handshaking is in effect. If PDQComm is unable to send data due to handshaking, it will set the CommEvent property to the error that occurred and will fire the OnComm event. If this happens, you may choose to repeatedly send the data until no error is reported, or disable handshaking. Keep in mind that if handshaking is disabled and the receiver is not ready to receive, data sent will be lost.

The default receive buffer size is 2048 bytes, which should be considered a minimum. If handshaking is employed, a larger receive buffer will not normally be needed. The receive buffer size can be increased up to 32K, if necessary, when handshaking cannot be used. The InBufferCount property indicates how many characters are present in the receive buffer and can be used to determine if the buffer is large enough. If the buffer is too small and an overflow occurs, the PDQComm control will set the CommEvent property to PDQ_ER_RXOVER and trigger the OnComm event.

# Terminal Emulations

Related Topics

The PDQComm control has terminal window and emulation functionality built into it. Terminal emulations allow you display text and to handle special control codes you receive over the serial port. These special control code may affect cursor position, screen colors, and other display attributes. Perhaps the best-known control codes on IBM computers are those recognized by the ANSI.SYS device driver.

PDQComm for Windows supports TTY, ANSI, VT-52, and VT-100 terminal emulations. Please consult the README.WRI file in the PDQComm product directory for the latest listing of the terminal emulations supported by PDQComm.

The following topics provide more information about using terminal windows and emulations for the PDQComm control:


Using PDQComm Control Terminal Emulations


Accessing the Clipboard in the Terminal Window


Accessing the Scroll-back Buffer


Scrolling Back While Receiving Data

# Using PDQComm Control Terminal Emulations

Using terminal emulations with the PDQComm control is very easy. All you have to do is set the Emulation property to the emulation you want. When you set the Disp property to a string of text that includes terminal codes (such as a string of data received from a BBS that sends ANSI codes), PDQComm interprets the codes and display the text correctly.

When Emulation is set to PDQ_EMULATION_NONE, the control is invisible at run time and iconic at design time. Although the control is invisible at run time, it will continue to process serial communications.

When Emulation is set to anything other than PDQ_EMULATION_NONE, the PDQComm control transforms into a terminal window, which will display text using the designated terminal emulation.

**NOTE** You can use any Windows font in the terminal window; however, consider using a fixed-width font, such as Terminal, to maintain row/column formatting in the terminal window. With a proportional font, such as Arial, character have variable widths and the text in the terminal window might appear to be unformatted. Also, if you want PDQComm to display IBM PC Characters (ASCII characters 128-255), then you must use a font, such as Terminal, that supports those characters. Terminal is the recommended DOS font, because it can display ASCII characters greater than 127.

---

One of the more important properties on the PDQComm control for control is the AutoSize property. When set to True, this property causes the terminal window to automatically size itself to full-screen, based on the selected font attributes (name, size, bold, etc.) and the size of the window as defined by the Rows and Columns properties.

The following properties define and control the behavior of a PDQComm terminal window:

| **Property** | **Description** |
|---|---|
| AutoScroll | Controls how the terminal window scrolls. |
| AutoSize | When True, the window sizes itself automatically. |
| BackColor | Background color (0-15). |
| BackSpace | Toggles destructive/non-destructive backspace key. |
| CaptureFilename | Opens and closes a text file for capture. |
| CaptureMode | Determines the way captured data is filtered. |
| Columns | Sets the width of the terminal window in columns. |
| CursorColumn | The current cursor column position. |
| CursorRow | The current cursor row position. |
| Disp | Used to display data in the terminal window. |
| Echo | Determines duplex. |
| Emulation | Determines terminal emulation. |
| Font*xxxx* | Standard Visual Basic Font properties. |
| ForeColor | Foreground color (0-15). |
| Rows | Sets the height of the terminal window in rows. |
| ScrollText | Contains the text in the scroll-back buffer. |
| ScrollRows | Determines the size of the scroll-back buffer. |
| Sel*xxx* | Standard VB SelText, SelStart, and SelEnd properties. |
| Text | Contains the text in the current terminal window. |

The PDQComm Control Reference contains a complete description for each of the PDQComm control properties listed in the previous table.

# Accessing the Clipboard in the Terminal Window

PDQComm terminal windows provide limited access to the Windows clipboard. The following table describes how a PDQComm terminal window supports the standard Windows cut and paste keystrokes:

| Keystroke | Description |
|---|---|
| CTRL-INS | Copies selected text to the clipboard |
| SHIFT-INS | Sends the text in the clipboard out the communications port. |
| SHIFT-DEL | Copies selected text to the clipboard in the same way CTRL-INS does. |

**NOTE** We do not allow the user to cut text out of the scroll-back buffer because there is no need to. The scroll-back buffer contains all the text received by PDQComm. Therefore, pressing SHIFT-DELhas the same effect as pressing CTRL-INS (Copy).

# Accessing the Scroll-back Buffer

The PDQComm custom control lets you allocate a scroll-back buffer to contain text that has been received. Here is a quick guide to using this important feature of PDQComm for Windows.

Assuming that terminal emulation is being used, and that the AutoScroll property has been set to allow the terminal window to scroll vertically, the first thing to consider is how many lines of text you want the scroll-back buffer to hold. You do this by setting the ScrollRows property to the number of lines desired. For example, to be able to scroll back 100 lines, set the ScrollRows property to 100:

```
Comm1.ScrollRows = 100
```

To use the scroll-back buffer, simply move the vertical scroll bar and the text will scroll back. To return to normal operations after you are done scrolling, press ESCAPE.

# Scrolling Back While Receiving Data

While data is being received, one of two of the following things may happen when you scroll back:

If you are using handshaking and you scroll back, PDQComm for Windows automatically signals the other PC to stop sending. You are free to browse the scroll-back buffer without losing characters, until you press ESCAPE. After pressing ESCAPE, PDQComm signals the other PC to continue sending.

If you are not using handshaking and scroll-back, PDQComm continues to display incoming data because it has no way to signal the other PC to stop sending data.

For this reason, handshaking should be used whenever possible with PDQComm for Windows.

# File Transfers

Related Topics

If you are new to file transfers, you must learn some simple terms.

*Download* — A verb that means to receive a file from another computer via modem using a binary file transfer protocol.

*Upload* — A verb that means to send a file to another computer via modem using a binary file transfer protocol.

*Protocol* — The method by which a file is transferred from one computer to another. A protocol is a strict set of rules that the two computers both agree to adhere to. One real-life protocol is the way people conduct themselves on the phone when calling for a friend.
**Answerer:** Hello?
**Caller:** Hi, this is Carl. Is Mike home?
**Answerer:** Hang on, Ill get him.
Although this is not as strict as a binary file transfer protocol, it is a protocol that most people observe. The only rule when using file transfer protocols is that both computers must be using the same protocol.

PDQComm supports the following file transfer protocols (as of this printing):

| | |
|---|---|
| XModem-CheckSum | YModem-G |
| XModem-CRC | ZModem |
| XModem-1K | Kermit |
| YModem-Batch | CompuServe B+ |

The PDQComm supports the following constants to represent the supported file transfer protocols in your code:

| | |
|---|---|
| PDQ_XMODEM_CHECKSUM | PDQ_YMODEM_G |
| PDQ_XMODEM_CRC | PDQ_ZMODEM |
| PDQ_XMODEM_1K | PDQ_KERMIT |
| PDQ_YMODEM_BATCH | PDQ_COMPUSERVE_BPLUS |

# Transferring Files with the PDQComm for Windows Control

Use the following general steps in your code to successfully transfer a file:

1. Pick a file transfer protocol and make sure both the sending and receiving systems use the same protocol. Use the XferProtocol property to set the protocol on your system.

2. Tell the receiving system to start the transfer.

3. To send a file (upload), set the Upload property to the filename that you want to send.

4. To receive a file (download), set the Download property to the filename that you want PDQComm to save the received file as.

5. As the file transfer commences, control returns to the application and the file transfer occurs in the background. The File Transfer Status window allows you to monitor the status of the transfer.

Once you set the Upload or Download properties, check the XferStatus property to get information about the transfer. If you want to constantly check the status, create a loop like this:

```
Download via Zmodem
 Comm1.XferStatusDialog = PDQ_XFERDIALOG_MODAL
 Comm1.XferProtocol = PDQ_ZMODEM
 Comm1.Download =

Wait until the transfer has started
Do
   Dummy = DoEvents()
Loop Until Comm1.XferStatus

This loop monitors the status of the file transfer
Do
   Dummy = DoEvents()
   CurStatus = Form1.Comm1.XferStatus
   If CurStatus = PDQ_XFER_TERM_OK Then
      Transfer was successful
      MsgBox Transfer Successful
      Exit Do
   ElseIf CurStatus = PDQ_XFER_TERM_ERROR Then
      Transfer was unsuccessful
      MsgBox Transfer Aborted
      Exit Do
   End If
Loop
```

**NOTE** Use DoEvents() to limit the system time taken away from the file transfer routine. Heavy activity in Windows or activity on a slower machine can cause time-out errors to occur and can terminate a file transfer.

When the file transfer is complete successfully without errors, the XferStatus property is set to PDQ_XFER_TERM_OK. When the file transfer aborts, the XferStatus property is set to PDQ_XFER_TERM_ERROR.

The PDQComm control creates and opens files on the local system in shared mode. This means that you can read incoming data as it is being received and written to a new file.

For more information on file transfers, see the FXFERBAS.WRI in the PDQComm product directory.

# BBS Programs

Related Topics

This topic contains a number of useful tips and techniques for implementing a BBS (Bulletin Board System) program. It does not serve as a complete tutorial on writing a full-featured BBS program. The HOST.VBP demonstration project, located in the PDQComm product directory, implements a simple host (dial-in) program that contains much of the same functionality that you would see in the BBS application. See this demonstration for more information.

Perhaps most important, when writing a BBS you can choose to echo back everything you receive. For example, when you log on to a commercial BBS, everything you type also appears on your screen. This is because the characters you enter first go from your computer to the BBS and then are sent back to your computer. Your computer displays everything it receives, which, of course, includes the characters you transmitted. The PDQComm control supports this behavior with the Echo property.

Another important feature of the PDQComm for Windows control is the Settings property, which lets you set and retrieve the current baud rate, parity, data bits, and stop bits. Because you may change the baud rate (or any other parameter) without closing the port, the Settings property will let you log in a caller at the callers baud rate, and then change your baud rate to match the callers.

To log in a caller, you simply wait for the word CONNECT, then read the baud rate (which always follows the word CONNECT). For example, if your modem returns CONNECT 2400 then the caller has connected at 2400 baud. You may then adjust your Settings property accordingly.

The following example demonstrates how to make your modem wait for a call, answer the phone, and connect to the callers modem at the callers baud rate:

```
Read all data on Input
Comm1.InputLen = 0

Tell the modem to answer the phone when the phone rings
InitString$ = ATS0=1 + Chr$(13)
Comm1.Output = InitString$

Flush the Input buffer
Comm1.InBufferCount = 0

This first loop executes until the two modems have connected
Do
   If Comm1.InBufferCount Then
      In$ = In$ + Comm1.Input
      C = Instr(In$, CONNECT)
      If C Then
         Left-trim the input string so it starts with CONNECT
         In$ = Mid$(In$, C)
         Bail out
         Exit Do
      End If
   End If
   Enable events
```

```
   Dummy = DoEvents()
Loop

This next loop executes until the connect baud rate is established
Do
   If Instr(In$, Chr$(13)) Then
      Carriage return
      Baud$ = Mid$(In$, C + 8)
      Exit Do
   End If
   Dummy = DoEvents()
   If Comm1.InBufferCount Then
      In$ = In$ + Comm1.Input
   End If
   Dummy = DoEvents()
Loop
Take off the trailing Carriage return
TrimBaud$ = RTrim$(Baud$)

Reset the baud rate
CurSettings$ = Comm1.Settings
TheRest$ = Mid$(CurSettings$, Instr(CurSettings$, ,))
NewSettings$ = TrimBaud$ + TheRest$
Comm1.Settings = NewSettings$
```

# Serial Port Communications Between Machines

Related Topics

One of the best features of the PDQComm control is its ability to read blocks of data in predefined chunks. Many industrial machines output data through a serial port in fixed-length blocks with no handshaking. This can sometimes makes it difficult for a PC to keep up with the machine. The PDQComm controls RThreshold property sets the threshold number of bytes that have to be received before PDQComm sets the CommEvent property to PDQ_EV_RECEIVE (received data) and fires the OnComm event. Therefore, if you want to read blocks of 10 bytes, just set RThreshold to 10, and the OnComm event will fire only after receive buffer receives 10 bytes. Thereafter, just read 10 bytes from the receive buffer.

# Using ModemWare

Related Topics

The idea behind ModemWare is that by adding a few files to your project, you can make high-level calls to procedures that automatically handle the intricacies of modems. All ModemWare routines the use the PDQComm control. You can use ModemWare routines to perform many common tasks, like place calls, send commands to a modem, wait for incoming data from a serial port, and more. ModemWare also provides the following:


Dialog boxes for configuring a serial port, terminal settings, and modem settings. Each of these dialogs writes their settings to a configuration file associated with your application. See the topic Control Port, Terminal, and Modem Settings for more information.


A database of modem configuration information with more than 440 entries for commercially available modems. See the topic Modem Database for more information.


A phone book which manages a list of entries and allows you to dial them. See the topic Phonebook Directory and Dialer for more information.


A powerful script language that allows non-programmers to record and edit scripts that accomplish such tasks as calling a host system, reading mail, and transferring files. See the topic ModemWare Scripts for more information.

# Control Port, Terminal, and Modem Settings

Related Topics

One of the most time consuming aspects of serial port programming is writing forms and code to configure the serial port and terminal. Most people don't even bother with modem setup because of the vast number of modems available, all of which have special requirements.

ModemWare contains the following routines that display dialog boxes for configuring a serial port, terminal settings, and modem settings: SetPortSettings, SetModemSettings, and SetTerminalSettings.

These dialogs write their settings to a configuration file. This file has the same name as your application, but with an INI extension, and is located in the Windows directory. For example, the following is the name of a configuration file for an application named PROJECT1:

```
C:\WINDOWS\PROJECT1.INI
```

Within an configuration file for an application, you can maintain separate setups for each PDQComm control in the application. A unique ID string in the Tag property for a PDQComm control provides the identifier for a setup sections for the control in the configuration file. For example, if Comm1 is used for receiving data only, set the Tag property of the control to Incoming. The heading of each section in the configuration file identifies the PDQComm control.

For example, the Port setup section for the Incoming port should look something like this:

```
[Port_Incoming]
Comm Port=2
Settings=19200,n,8,1
DTREnable=-1
RTSEnable=-1
Handshaking=3
Echo(Duplex)=0
```

And the Modem section might look like this:

```
[Modem_Incoming]
ModemNumber=136
ModemName=Hayes Ultra 14400 - Auto Reliable
HighestBaud=38400
InitString=AT&C1&D2W1X4S38=0S7=60S11=60^M
Connect=CONNECT
Attention=AT
Hangup=ATH^M
Reset=ATZ^M
Answer=ATA^M
Dial=ATDT
Busy=BUSY
```

The Terminal section might look like this:

```
[Terminal_Incoming]
Emulation=2
FontName=Terminal
```

```
FontSize=9
FontBold=0
BackColor=1
ForeColor=15
ScrollRows=409
```

ModemWare also supplies the following routines that load port, modem, and terminal settings for a PDQComm control from a configuration file: LoadPortSettings, LoadModemSettings, and LoadTerminalSettings.

# Modem Database

Related Topics

MODEMS.DAT is a data file that contains information for more than 440 modems. The information includes the modem's make and model, highest supported baud rate, initialization string, attention string, hangup string, busy string, dial string, and reset string. We at Crescent are constantly updating the MODEMS.DAT file with the latest configuration information for commercially available modems. You can also update and add modem entries to the MODEMS.DAT file using the utilities provided in the MWDEMO.VBP demonstration project.

Different ModemWare routines use the data in MODEMS.DAT to set modem settings for a PDQComm control. This means that you must distribute the MODEMS.DAT database with applications that use PDQComm and ModemWare to manage modems. At runtime, ModemWare routines look for this file in the same directory as your applications .EXE file. While using Visual Basic, ModemWare routines look for MODEMS.DAT in the project directory.

The first time a user runs your application, a dialog box with a list of modems appears and the user can pick from the list. The ModemWare dialog routine stores the modem information in the configuration file for the application. The next time the user runs the application, the application reads the modem information from the configuration file and configures the modem without displaying a dialog.

# Phonebook Directory and Dialer

Related Topics

With ModemWare, you can easily implement a phone book and dialer solution for your Visual Basic application. The MWPhoneBook routine displays a dialog box that allows users to create, view, and maintain a directory of phone numbers. For each entry, you can specify a name, phone number, baud rate, parity, data bits, stop bits, duplex (echo), handshaking, and a script file name. PDQComm saves all entries to the configuration file for the application.

The user can dial a selected entry in the phone book by clicking the Dial button in the dialog box. If there is a script file name specified for the dial routine, PDQComm executes the script. Otherwise PDQComm uses the standard dial routine, which continuously redials while the line is busy.

# ModemWare Scripts

Related Topics

ModemWare also provides several routines and a powerful scripting language that you can build into an application that allows users automate communications tasks. Users can record and edit scripts that call a host system, read mail, transfer files, and much more. This topic provides access to information about scripts and options for creating, developing, and maintaining them.


Defining a Script Task


Implementing a Script Using Visual Basic and ModemWare Routines


Writing a Script with a Text Editor and ModemWare Script Commands


Recording a Script with ModemWare Routines


Recording a Script with CTRLDEMO.VBP


Playing a Script File with ScriptPlay


Choosing Between Script Files and Visual Basic

# Defining a Script Task

Related Topics

The first thing you must do before you write a script is to define the task that you want the script to perform. For example, your task may be to call a host system, such as Compuserve, and download a file. You must manually log on to your host and make a note of all of the prompts that require you to enter information. You must also make a note of what you have to enter to complete the task. Write a script down on paper as you go.

For example, your script may look like this:

```
Dial the number.

Get: "Name:"
Send: "ModemWare"

Get: "Password:"
Send: "Script"

Get: "Menu:"
Send: "Download"

Get: "FileName:"
Send: "ALLFILES.ZIP"

Download the file.

Hangup.
```

It doesn't matter how you document the prompts and associated commands, as long as you can understand what text needs to be received and what text needs to be sent in response.

# Implementing a Script Using Visual Basic and ModemWare Routines

Related Topics

One way to implement a communications script is to use Visual Basic and the ModemWare WaitFor function. Visual Basic is the most powerful scripting language there is, so why not use it? This method gives you the most control; however, it does require more code and programming knowledge. By writing a script in Visual Basic and then compiling it into your application, you can also ensure the security of the information in the script.

The following Visual Basic code implements the sample script presented in the topic Defining a Script Task:

```
Sub Command1_Click()
   '-- ModemWare display flag.Tells WaitFor to display received data.
   MW_DisplayFlag = True
   '-- Data to be sent:
   MyName$ = "ModemWare"
   MyPassword$ = "Script"
   Download$ = "Download"
   FileName$ = "ALLFILES.ZIP"

   '-- WaitFor values:
   Timeout = 30 '-- Wait up to 30 seconds for each string.

   CD = True      '-- Reports if you've lost carrier.
   Ex = False     '-- more on this later.
   '-- Log in and tell the host to start a download.
   Wait$ = "Name:"
   Gosub GetTheData
   Comm1.Output = MyName$

   Wait$ = "Password:"
   Gosub GetTheData
   Comm1.Output = MyPassword$
   Wait$ = "Menu:"
   Gosub GetTheData
   Comm1.Output = Download$

   Wait$ = "FileName:"
   Gosub GetTheData
   Comm1.Output = FileName$

   '-- Download the file.

   '-- Use Zmodem.
   Comm1.XferProtocol = PDQ_ZMODEM

   '-- Display a non-modal dialog box.
   Comm1.XferStatusDialog = PDQ_XFERDIALOG_MODELESS
   '-- Start the download.
   Comm1.Download = ""
```

```
   '-- Wait until the transfer has started.
   Do
      Dummy = DoEvents()
   Loop Until Comm1.XferStatus PDQ_XFER_TERM_OK

   '-- Wait until the transfer has finished
   Do
      '-- Pause 1 second
      MWPause 1

      '-- Has the transfer ended yet?
      Status = Comm1.XferStatus
      If Status = PDQ_XFER_TERM_OK Then
         Exit Do
      Elseif Status = PDQ_XFER_TERM_ERROR Then
         Exit Do
      End If
   Loop
   '-- Hangup(2 attemps)
   Call Hangup( Comm1, 2, 1)
   '-- We're outta here.
   Exit Sub

   GetTheData:'-- Gosub that calls WaitFor$()
   '-- Add a carriage return.
   Wait$ = Wait$ & MW_CR$

   '-- Wait for the string.
   Received$ = WaitFor$( Comm1, Wait$, Timeout, CD, Ex )

   '-- What happened?
   If MWError() = MW_WAITFOR_OK Then
      MsgBox MWErrorMsg$()
      Exit Sub
   End If
   Return
End Sub
```

# Writing a Script with a Text Editor and ModemWare Script Commands

Related Topics

You can use a text editor to create and edit a script file containing ModemWare script commands. The following example is a text file containing the ModemWare script commands that implement the sample script presented in the topic Defining a Script Task:

```
PORT 2                '-- Use COM:
SETTINGS "2400,N,8,1"   '-- 2400 baud, no parity, 8 data bits,
                      '   1 stop bit.
TIMEOUT 60            '-- Wait up to 1 minute (WaitFor).

DIAL "555-1212"          '-- Dial the phone number.
WAITFOR "CONNECT"     '-- Wait until the modems connect.

PAUSE 3               '-- Pause 3 seconds to allow the
                      '   modems to completely connect.

WAITFOR "Name:"
SEND "ModemWare"

WAITFOR "Password:"
SEND "Script"

WAITFOR "Menu:"
SEND "download^M"

WAITFOR "FileName:"
SEND "ALLFILES.ZIP^M"

PAUSE 1
PROTOCOL "ZMODEM"'-- Use Zmodem.
DOWNLOAD'-- Download files.

HANGUP
END
```

The ModemWare ScriptPlay routine allows you to play ModemWare script commands specified in a text file from a Visual Basic application. See the topic Playing a Script File with ScriptPlay for more information about the ScriptPlay routine.

# Recording a Script with ModemWare Routines

Related Topics

You can save time writing script files by recording a script file with the ScriptRecord routine. When you record a script, ModemWare logs all of your keystrokes and received data, and writes a script file. You can then play the script to reproduce the recorded actions.

To record a script, you must load the following files in your project:

```
PDQCOM16.OCX, or PDQCOM32.OCX
COMDLG16.OCX or COMDLG32.OCX
COMMCTRL.BAS
MODMWARE.BAS
MWSCRIPT.BAS
```

You must also use terminal emulation (by setting the Emulation property of the comm control to 1, 2, 3, or 4).

To record a script, call the ScriptRecord routine with the following syntax:

```
Call ScriptRecord(Comm1, FileName$, Length%)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |
| *FileName$* | Specifies the file to receive the text of the recorded script. |
| *Length%* | Specifies how many characters to record when a WaitFor statement is recorded. |

To stop recording, call the ScriptRecord routine with the following syntax:

```
Call ScriptRecord (Comm1, "", 0)
```

The CTRLDEMO program is a great example of scripting techniques and the use of ModemWare scripting functionality.

# Recording a Script with CTRLDEMO.VBP

Related Topics

You can use the CTRLDEMO program that comes with PDQComm for Windows to record and play scripts.

**To record a script file with CTRLDEMO, do the following:**

1. Start the CTRLDEMO program.
2. Select the Record Script option from the Script menu.
3. Enter a filename for the script file in the file dialog box that appears.
4. Call the host system either manually by typing in ATDT *XXX-XXXX*(where *XXX-XXXX* is the phone number) or by selecting Phone Book from the Communications menu. If you selected the phone book option, you can add the host system to your phone book and click the Dial button. If for any reason you wish to abort the process at any time, you can press escape.
5. Once you've connected, do whatever actions you wish to record (i.e. download a file, etc) and then hangup.
6. Once you've completed the task or you simply wish to stop recording, select the Record Script option from the Script menu again to stop recording.

You can also use the options of the Script menu in CTRLDEMO to edit and play a script.

# Playing a Script File with ScriptPlay

Related Topics

Once you've written or recorded a script file, you can play it with the ModemWare ScriptPlay routine.

To play a script file, call the ScriptPlay routine with the following syntax:

```
Call ScriptPlay(Comm1, FileName$, ErrorLineNum%, ErrorLine$)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |
| *FileName$* | Specifies the file to play. |
| *ErrorLineNum%* | Returns the line number at which an error occurred (if any). |
| *ErrorLine$* | Returns the script line at which an error occurred (if any). |

# Choosing Between Script Files and Visual Basic

Related Topics

There are disadvantages to using ScriptPlay and ScriptRecord with script file. The first and most obvious disadvantage is that you don't have the power of the Visual Basic language at your disposal. Beyond that, script files are text files. Anyone can read your scripts. This can pose a potential problem if you need to keep your passwords and other information in the script file confidential. If you write your scripts usings Visual Basic, you can compile the script into an executable to ensure that the contents of the script is confidential.

The advantages of using script files are easy to see. You can modify your scripts with out using Visual Basic. Also, most non-programmers can easily understand the simple script language.

# ModemWare Tutorial - A Configurable Terminal Program

Related Topics

This tutorial shows you, step-by-step, how to create a working, configurable, powerful terminal program. It also helps you get your feet wet programming with PDQComm for Windows. The CTRLDEMO.VBP demonstration project, located in the PDQComm product directory, also implements a configurable terminal program. When you finish this tutorial, see the CTRLDEMO.VBP demonstration project for more information.

**To build your own configurable terminal program using the PDQComm control and ModemWare, do the following:**

1. Copy the Modem Database.
2. Load the ModemWare Files.
3. Place a PDQComm Control on Your Form.
4. Change the Status Message Display Code.
5. Add the Startup Code.
6. Add Exit Code.
7. Run the Program -- Last Minute Considerations.
8. Add a Setup Menu.
9. Add the Ability to Cancel Any Operation.
10. Add a Phonebook and Dialer.
11. Add File Transfer Capabilities.
12. Add File Transfer Monitoring Capabilites using the OnComm Event.

# Step 1 - Copy the Modem Database

Copy MODEMS.DAT from the PDQComm product directory to your Visual Basic product directory. ModemWare looks for this file in the same directory as your .EXE file. While using Visual Basic, the .EXE file is either VB.EXE or VB32.EXE, so be sure that MODEMS.DAT is in your VB directory.

# Step 2 - Load the ModemWare Files

Select Add File from Visual Basic's File menu and add the following files to a new project:

| Files | Description |
|---|---|
| PDQCOM32.OCX, PDQCOM16.OCX | PDQComm Control. |
| COMDLG32.OCX, COMDLG16.OCX | Crescent File Dialog Box used by ModemWare routines. |
| COMMCTRL.BAS | Contains declarations and constants for PDQComm for Windows, as well as some useful routines. |
| MODMWARE.BAS | Contains declarations and constants for ModemWare as well as all of the ModemWare routines. |
| MWPORT.BAS | Contains routines to set up the comm port. |
| MWPORT.FRM | Port setup dialog box. |
| MWTERM.BAS | Contains routines to set the Terminal. |
| MWTERM.FRM | Terminal setup dialog box. |
| MWMODEM.BAS | Contains routines to set up the Modem. |
| MWMODEM.FRM | Modem setup dialog box. |
| MWMODSTR.FRM | Modem strings setup dialog box. |
| MWSCRIPT.BAS | Contains scripting routines |

**NOTE** The PDQComm installation program installs .BAS, .FRM, and .FRX files for the 16-bit and 32-bit versions PDQComm for Windows product into separate product directories. These files are designed specifically for use with a 16-bit or 32-bit installation of Visual Basic 4.0 and should not be mixed into a single project.

##  Step 3 - Place a PDQComm Control on your Form

Double click the PDQComm control icon  in the Visual Basic ToolBox to place a PDQComm control on the current form.

## Step 4 - Change the Status Message Display Code

In the file, MODMWARE.BAS, go to the bottom of the MWStatus procedure. The last line is a call to a StatusPrint procedure, which displays a status message. Since this project has no StatusPrint procedure, you must create one. Add a module to your project, and create a StatusPrint subroutine using the following code:

```
Sub StatusPrint ( Msg$ )
   Form1.Caption = Msg$
End Sub
```

# Step 5 - Add the Startup Code

Place a Defint A-Z in the general declarations section of your form:

```
DefInt A-Z
```

In the Form_Load procedure, add the following code:

```
Sub Form_Load ()
   Me.Enabled = False

   Call LoadTerminalSettings ( Comm1 )
   Me.Show
   Me.Refresh

   Call LoadPortSettings ( Comm1 )
   Call LoadModemSettings ( Comm1 )

   Me.Enabled = True

   Comm1.AutoProcess = PDQ_AUTOPROCESS_BOTH

End Sub
```

This procedure uses the LoadTerminalSettings, LoadPortSettings, and LoadModemSettings routines to configure the terminal window, port, and modem associated with the PDQComm control.

It is a good idea to disable the form until everything has been set up. Although it may not be necessary, it is a good habit to disable your forms as they load. For this reason, we disable the form at the top of Form_Load, and enabled it at the bottom of Form_Load.

It is also a good idea to show your form as quickly as possible to give the user something to look at other than an hourglass. Since the terminal settings are the only settings that effect the way your program looks, you need to load the terminal settings first, then you can display your form with the Me.Show syntax. The Me.Refresh statement insures that the form is completely painted before moving on. When the form displays, the port and modem are initialized.

If you are creating a terminal (which we are), then set AutoProcess to PDQ_AUTOPROCESS_BOTH. This action automatically processes received data and places it in the terminal window. Also, keystrokes in the terminal window are sent out the port.

## Step 6 - Add Exit Code

The exit code is very simple. Place the following call in your Form_Unload event procedure:

```
Sub Form1_Unload ()
   Call MWEnd ( Comm1 )

   End

End Sub
```

The MWEnd procedure makes sure that the port closes correctly if it is open. Closing the port incorrectly could result in loss of transmitted data.

## Step 7 - Run the program: Last minute considerations

Before you run the program, here are two important considerations:

- ModemWare does support the Option Explicit Statement.
- You must set the start up form for the current project to Form1.

When you run this program for the first time, the first thing that comes up is the terminal setup dialog box. With it, you can change the emulation, font settings, colors, and number of scroll- back lines. There is also a color filter for supporting gray-scale or black and white monitors.

Click the OK button and the form will come up, followed by the port setup dialog box. With this dialog box, you can set the comm port, baud rate, parity, data bits, stop bits, and handshaking method. If an invalid port is specified, a message box will pop up with the message Invalid Port Selected

Click the OK button on the port dialog box, and the modem database dialog box will pop up. Select your modem from the list. If you can not find your modem listed you can do one of two things.

- Select a Hayes Compatible Modem.
- Add your Modem to the modem database by clicking the Add button. Enter your Modem's initialization string, and other information if necessary.

Once you've selected or added a Modem, click the Change button to initialize your Modem.

Close the program by double-clicking on the control button (in the upper left-hand corner of the form) or by clicking Visual Basic's Stop button. Now run the program again. Notice that all of your port, terminal, and modem settings are set, and the port is open.

The ModemWare setup routines store your port, terminal, and modem settings in a file in the \WINDOWS directory. The name of the file is derived from the name of your application, with an INI extension. For example, if this program is called PROJECT1.VBP, then your configuration file name is PROJECT1.INI. This file will always reside in the \WINDOWS directory. Go ahead and take a minute to look at it.

# Step 8 - Add a Setup Menu

Now that you have set up your application, how do you change the settings? Simply add a Setup menu to your application, with three menu options; Port, Terminal, and Modem. It should look something like this:

```
| Setup |
| Port |
| Terminal |
| Modem |
```

Underneath the Port menu option, use the following code to call the SetPortSettings routine:

```
Call SetPortSettings ( Comm1 )
```

Underneath the Terminal menu option, use the following code to call the SetTerminalSettings routine:

```
Call SetTerminalSettings ( Comm1 )
```

Underneath the Modem menu option, use the following code to call the SetModemSettings routine:

```
Call SetModemSettings ( Comm1 )
```

Run your program again. Now you have full control over the port, terminal, and modem settings.

# Step 9 - Add the Ability to Cancel Any Operation

You can cancel a script, dialing a phone number, or anything else, by simply adding one line of code to your Comm1_KeyPress event:

```
Sub Comm1_KeyPress ( KeyAscii As Integer )
   If KeyAscii = 27 Then MW_CancelFlag = True
End Sub
```

The MW_CancelFlag variable allows you to control the execution of the current ModemWare routine. From now on, anytime you press the escape key the current procedure will be canceled.

# Step 10 - Add a Phone Book and Dialer

Add the following files to your project:

| | |
|---|---|
| MWPHBOOK.BAS | Contains routines to access the phone book dialog box. |
| MWPHBOOK.FRM | Phone Book dialog box. |

Add a new menu option, Phone Book and use the following code in the menu option's click event procedure to call the MWPhoneBook routine:

```
Call MWPhoneBook ( Comm1 )
```

Run your program and select the Phone Book option. When the phone book dialog pops up, you will see one entry (for Crescent BBS). If you click the Dial button, you will be calling our bulletin board. If you want to add an entry, just click Add, enter the information, and click OK. If you change your mind in the middle of adding an entry and want to cancel, click Cancel.

You can delete an entry by clicking once on the entry, and then clicking Delete. For any entry, you can specify a script file in the Script Filename field. If you want to find a script file on your hard disk, click the button with the ellipses on it (...). All phone book entries are saved to your configuration file along with port, terminal, and modem settings.

# Step 11 - Add File Transfer Capabilities

To add the ability to transfer files, add the following files to your project:

| | |
|---|---|
| MWXFER.BAS | Contain Upload and Download routines. |
| MWXFER.FRM | Transfer Protocol dialog box (.OCX version) |

Then add two more menu options to your form: Upload and Download. Under the Upload menu option, use the following code to call the Upload routine:

```
Call Upload ( Comm1 )
```

Under the Download menu option, use the following code to call the Download routine:

```
Call Download ( Comm1 )
```

When the user selects Download or Upload the protocol dialog box appears, and the user can pick a protocol (Zmodem is the default). When the user picks a file name from the file dialog box the transfer will occur in the background. You can upload multiple files by selecting multiple files in the file dialog box.

## Step 12 - Add File Transfer Monitoring Capabilites using the OnComm Event

During a file transfer, PDQComm for Windows fires events to let you monitor the status of the transfer. You can capture these events in the OnComm event with the following code:

```
Sub Comm1_OnComm ()
   Select Case Comm1.CommEvent
      Case PDQ_EV_XFER
         Select Case Comm1.XferStatus
            Case PDQ_XFER_WAITING
               Msg$ = "Waiting For Next File"
            Case PDQ_XFER_FILE_READY
               Msg$ = "Preparing File"
            Case PDQ_XFER_FILE_START
               Msg$ = "Starting File Transfer"
            Case PDQ_XFER_XFERING
               Msg$ = "Transferring"
            Case PDQ_XFER_SKIP
               Msg$ = Skipping File"
            Case PDQ_XFER_ABORT
               Msg$ = "Transfer Aborted"
            Case PDQ_XFER_FINISHED
               Msg$ = "File Transferred"
            Case PDQ_XFER_LOSTCARRIER
               Msg$ = "Lost Carrier"
            Case PDQ_XFER_TIMEOUT
               Msg$ = "Transfer Timeout"
            Case PDQ_XFER_TERM_ERROR
               Msg$ = "Transfer Unsuccessful"
            Case PDQ_XFER_TERM_OK
               Msg$ = "Transfer Successful"
      End Select
      StatusPrint Msg$
   End Select
End Sub
```

By now, you should be well on your way to developing a configurable terminal application. See the CTRLDEMO.VBP demonstration project located in the PDQComm product directory for information and ideas on how to put the finishing touches on your application.

# PDQComm Control

Related Topics

The comm control embeds serial communications functionality into an application, allowing the application to transmit and receive data through a serial port.

**File Name**

PDQCOM32.OCX, PDQCOM16.OCX

**Class Name**

Comm

**Comments**

One comm control in an application manages a single serial port. You can use several comm controls in an application to manage several serial ports.

The comm control uses the Windows API and the Windows communications functions to handle serial communications. This means that PDQComm supports any serial port hardware that works with Windows. Use the Port utility of the Windows Control Panel to see port numbers and other information about the ports currently defined for Windows.

**Properties**

| | | |
|---|---|---|
| About | FontBold | ScrollRows * |
| AnswerBack * | FontItalic | ScrollText * |
| Appearance * | FontName | SelLength |
| AutoProcess * | FontSize | SelStart |
| AutoScroll * | FontUnderline | SelText |
| AutoSize * | ForeColor * | Settings * |
| AutoZModem * | Handshaking * | SmoothScroll * |
| BackColor * | Height | SThreshold * |
| BackSpace * | HelpContextID | TabIndex |
| Break * | hWnd | TabStop |
| CaptureFilename * | InBufferCount * | Tag |
| CaptureMode * | InBufferSize * | Text * |
| CDHolding * | Index | Top |
| ColorFilter * | Input * | Upload * |
| Columns * | InputLen * | Visible |
| CommEvent * | Interval * | WhatsThisHelpID |
| CommID * | InTimeout * | Width |
| CommPort * | KeyTranslation * | XferCarrierAbort * |
| Container | Left | XferDestFilename * |
| CTSHolding * | LineInput * | XferDialogHeight * |
| CursorColumn * | Name | XferDialogLeft * |
| CursorRow * | NullDiscard * | XferDialogTop * |
| CursorType * | Object | XferDialogWidth * |

| | | |
|---|---|---|
| Disp * | OutBufferCount * | XferFileDate * |
| Download * | OutBufferSize * | XferFileSize * |
| DragIcon | Output * | XferFileTime * |
| DragMode | Parent | XferMessage * |
| DSRHolding * | ParityReplace * | XferProtocol * |
| DTREnable * | PortOpen * | XferSourceFilename * |
| Echo * | Rows * | XferStatus * |
| Emulation * | RThreshold * | XferStatusDialog * |
| Font | RTSEnable * | XferTransferred * |

* A custom or modified property, method, or event.

**Methods**

| | | |
|---|---|---|
| CloseCommHandle * | Move | ShowWhatsThis |
| CRC16 * | OpenCommHandle * | SuspendComm * |
| CRC32 * | SelFocus | ZOrder |
| Drag | | |

**Events**

OnComm *

**Constants**

Several properties of the PDQComm control take enumerated values. These properties can only accept values from a discrete list of integer values. Each integer value represents a behavior setting for the property. To make your code more readable, the PDQComm control supports the use of constants for reading and setting the value of enumerated properties. The PDQComm .OCXs automatically establish constants for enumerated property values.

# AnswerBack Property

The AnswerBack property contains a string that you want PDQComm to send to a remote system in response to an ENQ character.

### Syntax

**[***form***.]***Comm1*.AnswerBack**[** = *string$***]**

### Data Type

String

### Usage

Read/Write at design time and runtime.

### Comments

The ENQ character is ASCII character 5 ( Chr$(5) ). The ENQ character is usually used for identifying a terminal. If the AnswerBack property is not null, PDQComm emulation responds with the AnswerBack string upon receiving an ENQ. If AnswerBack is null, PDQComm ignores the ENQ.

# Appearance Property

The Appearance property controls whether or not the PDQComm emulation has the Windows 95 3D appearance.

**Syntax**

**[***form.***]***Comm1*.Appearance**[** = *integer%***]**

**Data Type**

Integer

**Usage**

Read/Write at design time only.

**Comments**

Valid settings for this property are:

| Value | Description |
|---|---|
| 0 | Flat - The emulation has the normal, non-3D appearance. |
| 1 | 3D - The emulation has the Windows 95 3D appearance. |

If the Emulation property is set to 0 (no emulation), set the Appearance property to Flat to suppress the appearance of a 3D border around your control. The 3D border causes PDQComm to take more screen space.

**NOTE** This property has no effect on the Microsoft Windows NT 3.51 platform or earlier platforms.

# AutoProcess Property

The AutoProcess property allows you to automate the processing of serial port and keyboard data. AutoProcess provides a direct link between the serial port and the terminal window of the comm control.

### Syntax

**[***form.***]***Comm1*.AutoProcess**[** = *integer%***]**

### Data Type

Integer

### Usage

Read/Write at design time and runtime.

### Comments

Valid settings for this property are:

| Constant | Value | Description |
|---|---|---|
| PDQ_AUTOPROCESS_NONE | 0 | The comm control does not automatically process serial or keyboard input. If AutoProcess is set to 0, the PDQ_EV_RECEIVE event will not fire if RThreshold is 0. See the RThreshold property and the OnComm event for more information. |
| PDQ_AUTOPROCESS_SERIAL | 1 | All data that is received by the serial port is automatically displayed in the terminal window (assuming that the Emulation property is non-zero). |
| PDQ_AUTOPROCESS_KEY | 2 | All keys pressed in the terminal window are sent out the serial port (assuming that the Emulation property is non-zero). |
| PDQ_AUTOPROCESS_BOTH | 3 | PDQComm acts as a terminal program, both displaying data that comes in over the serial port and also sending keystrokes out the serial port. |

**NOTE** When AutoProcess is in use, PDQ_EV_RECEIVE events do not fire the OnComm event. Also, do not directly access the receive and transmit buffers (i.e., Input, Output, and LineInput properties). If you need to directly access the receive and transmit buffers, disable AutoProcess while accessing the buffers, then re-enable AutoProcess when you are done.

# AutoScroll Property

The AutoScroll property controls automatic scrolling behavior for the terminal window of the comm control.

**Syntax**

**[***form*.**]***Comm1*.AutoScroll**[** = *integer%***]**

**Data Type**

Integer

**Usage**

Read/Write at design time and runtime.

**Comments**

Valid settings for this property are:

| Constant | Value | Description |
|---|---|---|
| PDQ_AUTOSCROLL_NONE | 0 | The terminal window does not scroll automatically. |
| PDQ_AUTOSCROLL_VERTICAL | 1 | The terminal window scrolls vertically when the cursor goes beyond the bottom-most row. |
| PDQ_AUTOSCROLL_HORIZONTAL | 2 | The terminal window scrolls horizontally when the cursor goes beyond the right-most column |
| PDQ_AUTOSCROLL_BOTH | 3 | The terminal window scrolls vertically when the cursor goes beyond the bottom-most row, and horizontally when the cursor goes beyond the right-most column |
| PDQ_AUTOSCROLL_VERTKEY | 4 | The terminal window scrolls vertically when the cursor goes beyond the bottom-most row, and only scrolls horizontally if you type beyond the right-most column. |

AutoScroll only affects PDQComm when AutoSize is False and the visible number of rows and/or columns is less than the number specified by the Rows and Columns properties

# AutoSize Property

The AutoSize property controls automatic sizing behavior for the terminal windows of the comm control.

**Syntax**

**[***form***.]***Comm1*.AutoSize**[** = *boolean%***]**

**Data Type**

Boolean Integer

**Usage**

Read/Write at design time and runtime.

**Comments**

If AutoSize is True, the terminal window automatically sizes itself to the number of rows and columns specified by the Rows and Columns properties, taking into account the size of the current font. If AutoSize is False, the terminal window retains the same size, as drawn on the Visual Basic form.

**NOTE** If you are using a proportional font, the width of the terminal window may be less than the width of some lines of text. This occurs because PDQComm calculates the terminal width to be the average character width of the font multiplied by the number of columns as specified by the Columns property.

# AutoZModem Property

The AutoZModem property controls whether the comm control examines the incoming data stream for a Zmodem-receive startup sequence.

### Syntax

**[***form*.**]***Comm1*.AutoZModem**[** = *boolean%***]**

### Data Type

Boolean Integer

### Usage

Read/Write at design time and runtime.

### Comments

If AutoZModem is True, the comm control examines the incoming data stream for the Zmodem-receive startup sequence. If the sequence is found, the comm control fires the PDQ_EV_ZMODEM event. You can write an event procedure for the OnComm event that initiates a Zmodem download. If you do not need to process the Zmodem-receive startup sequence, set the AutoZModem property to False. This disables the examination of the incoming data stream and enhances performance.

# BackColor Property

The BackColor property sets and returns the background color of the terminal window of the comm control.

**Syntax**

**[***form***.]***Comm1***.BackColor[** = *integer%***]**

**Data Type**

Integer

**Usage**

Read/write at design time and runtime.

**Comments**

Valid settings for this property are:

| Color | Setting Value | RGB Value (Hex) |
|---|---|---|
| PDQ_COLOR_BLACK | 0 | &H000000 |
| PDQ_COLOR_BLUE | 1 | &H800000 |
| PDQ_COLOR_GREEN | 2 | &H008000 |
| PDQ_COLOR_CYAN | 3 | &H808000 |
| PDQ_COLOR_RED | 4 | &H000080 |
| PDQ_COLOR_MAGENTA | 5 | &H800080 |
| PDQ_COLOR_YELLOW | 6 | &H008080 |
| PDQ_COLOR_WHITE | 7 | &HC0C0C0 |
| PDQ_COLOR_GRAY | 8 | &H808080 |
| PDQ_COLOR_LIGHTBLUE | 9 | &HFF0000 |
| PDQ_COLOR_LIGHTGREEN | 10 | &H00FF00 |
| PDQ_COLOR_LIGHTCYAN | 11 | &HFFFF00 |
| PDQ_COLOR_LIGHTRED | 12 | &H0000FF |
| PDQ_COLOR_LIGHTMAGENTA | 13 | &HFF00FF |
| PDQ_COLOR_LIGHTYELLOW | 14 | &H00FFFF |
| PDQ_COLOR_BRIGHTWHITE | 15 | &HFFFFFF |

**NOTE** The BackColor property accepts only the values 0 through 15. It does not accept standard Visual Basic RGB values. The colors represented by the values 0 through 15 maintain compatibility with terminal emulations designed for DOS text-mode.

# BackSpace Property

The BackSpace property controls the behavior of BackSpace characters in the terminal window of the comm control.

### Syntax

**[***form.***]***Comm1*.BackSpace**[** = *integer%***]**

### Data Type

Integer

### Usage

Read/Write at design time and runtime.

### Comments

Valid settings for this property are:

| Constant | Value | Description |
|---|---|---|
| PDQ_BACKSPACE_DESTRUCTIVE | 0 | Backspace characters are destructive, and erase the previous character in the terminal window (default). |
| PDQ_BACKSPACE_NON_DESTRUCTIVE | 1 | Backspace characters are non-destructive, and do not erase the previous character in the terminal window. |

# Break Property

The Break property allows you to set or clear a break signal.

### Syntax

**[***form***.]***Comm1*.Break**[** = *boolean%***]**

### Data Type

Boolean Integer

### Usage

Write only at runtime only.

### Comments

When you set the Break Property to True (-1), the comm control sends a break signal. When set to False (0), the comm control clears the break signal. You can use this property with the Visual Basic Timer control to control the duration of the break signal. See the documentation for the Visual Basic Timer control for more information on timing critical events.

### Example

The following example shows how to send a Break signal for a tenth of a second:

```
'--- Set the Break condition
Comm1.Break = True
'--- Set duration to a 1/10 second
Duration! = Timer + .1
'--- Wait for the duration to pass
Do Until Timer > Duration!
   Dummy = DoEvents()
Loop
'--- Clear the Break condition
Comm1.Break = False
```

# CaptureFilename Property

When the CaptureFilename is set to a valid filename PDQComm will initiate data capture, which saves all data that comes over the serial port to the filename specified. The way in which the data is filtered is determined by the CaptureMode property.

**Syntax**

`[`*form*`.]`*Comm1*`.CaptureFilename[ = `*filename$*`]`

**Data Type**

String

**Usage**

Write only at runtime only.

**Comments**

Capture mode is a term used in terminal program software. It simply lets you save whatever data you receive to a file.

The capture file is opened in shared mode, allowing your program to open it for read only. However, you can not write to the file until the file has been closed, by setting the CaptureFilename to an empty string (). If the file already exists, new data will be apended to it.

# CaptureMode Property

The CaptureMode property sets and returns the capture mode, which determines how data is filtered before being saved to the file specified by the CaptureFilename property.

### Syntax

**[***form.***]***Comm1*.CaptureMode**[** = *integer%***]**

### Data Type

Integer

### Usage

Read/Write at design time and runtime.

### Comments

Capture mode is a term used in terminal program software. It simply lets you save whatever data you receive to a file.

Valid settings for this property are:

| Constant | Value | Description |
|---|---|---|
| PDQ_CAPTURE_STANDARD | 0 | Data is saved without color codes in the exact order that it was received. For example, you could receive an ANSI string to draw a line at row 20, followed by an ANSI string to draw a line at row 19. The capture file would contain row 20 followed by row 19, not the way it appears on the screen |
| PDQ_CAPTURE_BINARY | 1 | Nothing is filtered and data is saved verbatim. |
| PDQ_CAPTURE_VISIBLE | 2 | Only the text is saved and written to disk, exactly as it looks in the terminal window (without the color codes). Because of the way PDQ_CAPTURE_VISIBLE works, data is required to scroll off the screen, to be saved to the file. If the incoming data is sending control codes to erase the screen with scrolling, data will not be saved to the capture file. |

See the CaptureFilename property for information on capturing received data.

# CDHolding Property

The CDHolding property returns True if the Carrier Detect (CD) line is high (if a carrier is present).

**Syntax**

`[`*boolean%* `= ][`*form*`.]`*Comm1*`.CDHolding`

**Data Type**

Boolean Integer

**Usage**

Read only at runtime only.

**Comments**

It is especially important to trap a loss of carrier in a host-remote application such as a bulletin board, because the caller can hang up (dropping the carrier) at any time. This should be done by monitoring the PDQ_EV_CD event in OnComm and checking the CDHolding property.

**NOTE** Some modems and cables do not implement the Carrier Detect line. The Carrier Detect is also known as the Receive Line Signal Detect (RLSD).

# ColorFilter Property

The ColorFilter property sets and returns the color mode of the terminal window.

### Syntax

**[***form***.]***Comm1***.ColorFilter[** = *integer%***]**

### Data Type

Integer

### Usage

Read/Write at design time and runtime.

### Comments

Valid settings for this property are:

| Constant | Value | Description |
|---|---|---|
| PDQ_COLOR_FULL | 0 | The BackColor and ForeColor are active in full color. |
| PDQ_COLOR_GREY | 1 | All color values are scaled to gray. |
| PDQ_COLOR_MONO | 2 | All color values are scaled to the BackColor and ForeColor that are currently set. You would use gray scaled or monochrome colors if, for example, you are running on a laptop with 64-shade VGA video hardware. |

# Columns Property

The Columns property sets and returns the number of columns in the terminal emulator.

**Syntax**

**[***form***.]***Comm1***.**Columns**[** = *integer%***]**

**Data Type**

Integer

**Usage**

Read/Write at design time and runtime.

**Comments**

The Columns property does not refer to the width of the control. Rather, it refers to the number of fixed-pitch columns for the terminal emulation, just as a standard DOS text screen has 80 columns.

**NOTE** When Emulation is set to PDQ_EMULATION_NONE (iconic), the Columns property is ignored.

# CommEvent Property

The CommEvent property returns the most recent communications event or error.

**Syntax**

`[integer% = ][form.]Comm1.CommEvent`

**Data Type**

Integer

**Usage**

Read only at runtime only.

**Comments**

Although the OnComm event occurs whenever a communications error or event occurs, the CommEvent property holds the code number for the error or event. It is therefore necessary to read the CommEvent property once the OnComm event is occurs in order to determine the actual error or event.

Valid code numbers returned by CommEvent are:

**Communications Events:**

| Constant | Value | Description |
|---|---|---|
| PDQ_EV_SEND | 1 | There are fewer than SThreshold number of characters in the output buffer |
| PDQ_EV_RECEIVE | 2 | Received RThreshold number of characters. |
| PDQ_EV_CTS | 3 | Change in Clear To Send line. |
| PDQ_EV_DSR | 4 | Change in Data Set Ready line. |
| PDQ_EV_CD | 5 | Change in Carrier Detect line. |
| PDQ_EV_RING | 6 | Ring detected. |
| PDQ_EV_EOF | 7 | End Of File (ASCII character 26) character received. |
| PDQ_EV_ZMODEM | 8 | ZModem startup sequence. |
| PDQ_EV_XFER | 100 | File transfer event. Read the XferStatus property for the current file transfer status. |

**Communications Errors:**

| Constant | Value | Description |
|---|---|---|
| PDQ_ER_BREAK | 1001 | A Break signal was received. |
| PDQ_ER_CTSTO | 1002 | Clear To Send Timeout. The CTS line was low for CTSTimeout number of milliseconds while trying to transmit a character. CTSTimeout is an obsolete property. |
| PDQ_ER_DSRTO | 1003 | Data Set Ready Timeout. The DSR line was low for DSRTimeout number of milliseconds while trying to transmit a character. DSRTimeout is an obsolete property. |
| PDQ_ER_FRAME | 1004 | Framing Error. The hardware detected a framing error. Generally, you cannot do anything about this error, except to try to send |

| | | |
|---|---|---|
| | | or receive again. |
| PDQ_ER_INTO | 1005 | Input Timeout. The requested number of characters specified by the InputLen property could not be returned by the Input or LineInput properties before InTimeout number of milliseconds have passed. |
| PDQ_ER_OVERRUN | 1006 | Port Overrun. A character was not read from the hardware before the next character arrived, and was thus lost. If you receive this error, try lowering your baud rate or upgrading your serial port hardware. |
| PDQ_ER_CDTO | 1007 | Carrier Detect Timeout. The CD line was low for CDTimeout number of milliseconds while trying to transmit a character. Carrier Detect is also known as the Receive Line Signal Detect (RLSD). CDTimeout is an obsolete property. |
| PDQ_ER_RXOVER | 1008 | Receive Buffer Overflow. There is no room in the input buffer. |
| PDQ_ER_RXPARITY | 1009 | Parity Error. The hardware detected a parity error. |
| PDQ_ER_TXFULL | 1010 | Transmit Buffer Full. The output buffer is full while trying to queue a character. |

The PDQComm OCXes automatically establish constants for these values.

# CommID Property

The CommID property returns the communications device identification number.

**Syntax**

`[integer% = ][form.]Comm1.CommID`

**Data Type**

Integer

**Usage**

Read/Write at runtime only.

**Comments**

The value of the CommID property identifies the instance handle of the control's open communications port. This is the same value that is used by the Windows internal communications routines. Use this property for a communications DLL routine that expects a valid handle to an open communications port to be passed as an argument.

You can also set this property with a valid instance handle from another program. The PortOpen property is False before setting this property. When you set this property, the appropriate properties of the comm control are set to the current port settings and the PortOpen property is True.

# CommPort Property

The CommPort property sets and returns the communications port number.

### Syntax

**[***form***.]***Comm1***.CommPort[** = *integer%***]**

### Data Type

Integer

### Usage

Read/Write at design time and runtime.

### Comments

If the specified port is invalid, PDQComm generates the Visual Basic "Device Unavailable" error (#68), causing an error dialog box to pop up. To prevent this from happening, use On Error when you open the port. See the example below for more information.

**NOTE** It is very important that you set the CommPort property before opening the port.

### Example

```
'-- The following example traps an error 68 caused by opening
'-- an unavailable communications port.

On Error Resume Next

Do
   In$ = InputBox$("Enter comm Port:")
   PortNum = Val(In$)
   If PortNum Then
      Comm1.CommPort = PortNum
      Comm1.PortOpen = True
      If Err Then
         MsgBox "Port is invalid, try again"
      End If
   Else
      Exit Do
   EndIf
Loop
```

# CTSHolding Property

The CTSHolding property returns True if the Clear To Send line is high.

**Syntax**

`[`*boolean%* `= ][`*form*`.]`*Comm1*`.CTSHolding`

**Data Type**

Boolean Integer

**Usage**

Read only at runtime only.

**Comments**

The CTS line is used in RTS/CTS hardware handshaking. The CTSHolding property gives you a way to manually poll the CTS line.

For more information on handshaking protocols, see the Handshaking property description and the topic Handshaking and the Receive Buffer.

# CursorColumn Property

The CursorColumn property sets and returns the current cursor column position (1 = first column).

### Syntax

**[***integer% =* **][***form.***]***Comm1*.CursorColumn

### Data Type

Integer

### Usage

Read/Write at runtime only.

### Comments

Column coordinates range from 1 to the number of columns specified by the Columns property.

# CursorRow Property

The CursorRow property sets and returns the current cursor row position (1 = top row).

### Syntax

**[***integer% =* **][***form.***]***Comm1*.CursorRow

### Data Type

Integer

### Usage

Read/Write at runtime only.

### Comments

Row coordinates range from 1 to the number of row specified by the Rows property.

# CursorType Property

The CursorType property sets and returns the type of cursor displayed.

### Syntax

**[***form*.**]***Comm1*.CursorType**[** = *integer%***]**

### Data Type

Integer

### Usage

Read/Write at design time and runtime.

### Comments

The CursorType property can have one of the following values:

| Constant | Value | Description |
|---|---|---|
| PDQ_CURSOR_VBAR | 0 | Vertical bar cursor |
| PDQ_CURSOR_BLOCK | 1 | Block cursor |

The default value is PDQ_CURSOR_BLOCK.

# Disp Property

The Disp property writes the specified string to the current terminal emulation.

**Syntax**

**[***form***.]***Comm1***.Disp[** *= string$***]**

**Data Type**

String

**Usage**

Write only at runtime only.

**Comments**

Disp displays text at the current cursor position in the terminal emulation. After displaying text with Disp, the cursor position is reset to the end of the displayed text.

**NOTE** When you display a string with the Disp property, the string is only sent to the terminal window, not out the communications port.

You can display more than one line of text using Disp by embedding Chr$(13) & Chr$(10) characters in the string.

# Download Property

The Download property initiates a binary file transfer to download (receive) a file, or appends a file to the current download queue.

### Syntax

**[***form.***]***Comm1*.Download**[** = *filename$***]**

### Data Type

String

### Usage

Read/Write at runtime only.

### Comments

When the Download property is set to a string that contains a valid filename (or a null string for Batch downloads), PDQComm starts a download operation using the file transfer protocol defined by the XferProtocol property.

Immediately after setting the Download property, PDQComm returns control to your program. The transfer occurs in the background. You can monitor the status of the transfer with the XferStatus property. You can also display the file transfer status using the built in file transfer dialog box. See the XferStatusDialog property for more information.

With batch protocols such as YModem-Batch, you can assign an empty string to the Download property. This causes PDQComm to read filenames from the file header sent by the sending computer.

If you set the filename to an empty string () when using a non-batch protocol, such as XModem, PDQComm generates a filename called UNTITLED.XXX, where XXX is an incremental number. PDQComm first attempts to create UNTITLED.000. If that file exists, PDQComm increments the filename to UNTITLED.001, and so on until it can generate a unique filename.

**NOTE** When XferStatus is greater than 0 (file is being transferred), you cannot access the CommPort, Input, Output, LineInput, Settings or PortOpen properties, or set the InBufferCount and OutBufferCount properties.

See the topic File Transfers for more information.

See the Download procedure in CTRLDEMO.BAS which is used in CTRLDEMO.MAK, the PDQComm for Windows custom control demo terminal program.

### Examples

The following example will start an XModem download using the built-in file transfer status dialog box:

```
'-- Set the protocol to XModem-Checksum
Comm1.XFerProtocol = PDQ_XMODEM_CHECKSUM
'-- Use the built-in dialog box to display status
    information
Comm1.XferStatusDialog = PDQ_XFERDIALOG_MODAL
'-- Download the file
Comm1.Download = "FILE1.ZIP"
```

```
'-- Control immediately comes back here after setting
'   the Upload property, and the file transfer will happen
'   automatically in the background.
```

The following code will start a YModem-Batch download using the built-in file transfer status dialog box:

```
'-- Set the protocol to YModem-Batch
Comm1.XFerProtocol = PDQ_YMODEM_BATCH
'-- Use the built-in dialog box to display status information
Comm1.XferStatusDialog = PDQ_XFERDIALOG_MODAL
'-- Download the files
Comm1.Download = ""
'-- Control immediately comes back here after setting the
'   Download property, and the file transfer will happen
'   automatically in the background.
```

Always call DoEvents if you enter any loops after starting a file transfer to allow the system time to perform the transfer.

# DSRHolding Property

The DSRHolding property returns True if the Data Set Ready line is high.

### Syntax

**[***boolean%* **=** **][***form***.]***Comm1*.DSRHolding

### Data Type

Boolean Integer

### Usage

Read only at runtime only.

### Comments

The DSR line can be used to perform hardware handshaking. The DSRHolding property provides a way to manually poll the DSR line.

# DTREnable Property

The DTREnable property toggles the state of the DTR line.

### Syntax

**[***form***.]***Comm1***.**DTREnable**[ = ***boolean%***]**

### Data Type

Boolean Integer

### Usage

Read/write at design time and runtime.

### Comments

When DTREnable is set to True, the Data Terminal Ready line is set high (True) if the port is open or low (False) if the port is closed. When DTREnable is set to False the DTR line always remains low.

# Echo Property

The Echo property indicates whether characters that are transmitted should be echoed on the terminal screen.

**Syntax**

**[***form*.**]***Comm1*.Echo**[** = *boolean%***]**

**Data Type**

Boolean Integer

**Usage**

Read/write at design time and runtime.

**Comments**

In communications terms, the Echo property controls the duplex of the communications mode. If Echo is set to False (Full Duplex), character transmitted using the Output property get sent out the port. However, if Echo is True (Half Duplex), characters transmitted using the Output property get sent out of the port and then display on the terminal window.

When communicating with a BBS or any other online service, set Echo to false because the BBS automatically echoes back every character it receives. However, when communicating modem-to-modem in chat mode (two terminal programs connected via a modem), set Echo to True because terminal programs do not echo received data.

# Emulation Property

The Emulation property specifies the current terminal emulation.

### Syntax

**[***form***.]***Comm1***.Emulation[** = *integer%***]**

### Data Type

Integer

### Usage

Read/write at design time and runtime.

### Comments

Valid setting for this property are:

| Constant | Value | Description |
|---|---|---|
| PDQ_EMULATION_NONE | 0 | The control is invisible at run-time and iconic at design-time. Although the control is invisible at runtime, it will continue to process serial communications. |
| PDQ_EMULATION_TTY | 1 | Emulation window handles standard DOS control codes but does not support color or cursor position sequences. |
| PDQ_EMULATION_ANSI | 2 | Supports ANSI escape sequences allowing for color and cursor positioning. |
| PDQ_EMULATION_VT52 | 3 | Supports VT52 escape sequences allowing for color and cursor positioning. |
| PDQ_EMULATION_VT100 | 4 | Supports VT100 escape sequences allowing for color and cursor positioning. |

The default value is PDQ_EMULATION_NONE.

# ForeColor Property

The ForeColor property sets and returns the foreground color of the terminal window of the comm control.

**Syntax**

**[***form***.]***Comm1***.ForeColor[** = *integer%***]**

**Data Type**

Integer

**Usage**

Read/write at design time and runtime.

**Comments**

Valid settings for this property are:

| Color | Setting Value | RGB Value (Hex) |
|---|---|---|
| PDQ_COLOR_BLACK | 0 | &H000000 |
| PDQ_COLOR_BLUE | 1 | &H800000 |
| PDQ_COLOR_GREEN | 2 | &H008000 |
| PDQ_COLOR_CYAN | 3 | &H808000 |
| PDQ_COLOR_RED | 4 | &H000080 |
| PDQ_COLOR_MAGENTA | 5 | &H800080 |
| PDQ_COLOR_YELLOW | 6 | &H008080 |
| PDQ_COLOR_WHITE | 7 | &HC0C0C0 |
| PDQ_COLOR_GRAY | 8 | &H808080 |
| PDQ_COLOR_LIGHTBLUE | 9 | &HFF0000 |
| PDQ_COLOR_LIGHTGREEN | 10 | &H00FF00 |
| PDQ_COLOR_LIGHTCYAN | 11 | &HFFFF00 |
| PDQ_COLOR_LIGHTRED | 12 | &H0000FF |
| PDQ_COLOR_LIGHTMAGENTA | 13 | &HFF00FF |
| PDQ_COLOR_LIGHTYELLOW | 14 | &H00FFFF |
| PDQ_COLOR_BRIGHTWHITE | 15 | &HFFFFFF |

**NOTE** The ForeColor property accepts only the values 0 through 15. It does not accept standard Visual Basic RGB values. The colors represented by the values 0 through 15 maintain compatibility with terminal emulations designed for DOS text-mode.

# Handshaking Property

The Handshaking property sets and returns the hardware handshaking protocol.

### Syntax

**[***form***.]***Comm1***.Handshaking[** = *integer%***]**

### Data Type

Integer

### Usage

Read/write at design time and runtime.

### Comments

The term handshaking refers to the protocol by which data is transferred from the hardware port to the receive buffer. This process happens under the hood and is invisible to the programmer. When a byte of data arrives at the serial port, the communications device has to move it into the receive buffer so that your program can read it. If there was no receive buffer and your program attempted to read every byte directly from the hardware, you would lose characters. A handshaking protocol ensures that data is not lost due to a buffer overrun, in which data arrives at the port too quickly for the communications device to move the data into the receive buffer.

| Constant | Value | Description |
|---|---|---|
| PDQ_HANDSHAKING_NONE | 0 | Turns off handshaking. Beware of possible buffer overruns. |
| PDQ_HANDSHAKING_XON | 1 | Turns on software handshaking. Control codes are sent between systems to control the flow of data. |
| PDQ_HANDSHAKING_RTS | 2 | Turns on hardware handshaking. Use RTS handshaking when communicating with a high speed modem. |
| PDQ_HANDSHAKING_BOTH | 3 | Turns on hardware and software handshaking. See the descriptions of the previous two values for information. |

See the topic Handshaking and the Receive Buffer for more information on handshaking and why it is used.

# InBufferCount Property

The InBufferCount property returns the number of characters waiting in the receive buffer.

### Syntax

```
[form.]Comm1.InBufferCount[ = integer%]
[integer% = ][form.]Comm1.InBufferCount
```

### Data Type

Integer

### Usage

Read/write at runtime only.

### Comments

InBufferCount refers to the number of characters that have been received by the modem and are waiting in the input buffer for processing. You can flush the input buffer by setting the InBufferCount property to zero.

Do not confuse this property with InBufferSize, which reflects the total size of the input buffer.

### Example

This example shows how to poll for received data:

```
'--- Specifies to receive all available data
Comm1.InputLen = 0
Do
   '--- Allow other processes to continue
   Dummy = DoEvents()
   '--- Check for data
   If Comm1.InBufferCount Then
      '--- Read data
      In$ = Comm1.Input
   End If
Loop
```

# InBufferSize Property

The InBufferSize property sets and returns the size of the input buffer in bytes.

**Syntax**

**[***form***.]***Comm1***.**InBufferSize**[ = ***integer%***]**

**Data Type**

Integer

**Usage**

Read/write at design time and runtime.

**Comments**

InBufferSize refers to the total size of the input buffer. The default size is 4096 bytes.Do not confuse this property with InBufferCount, which reflects the number of bytes currently waiting in the input buffer.

**NOTE** If handshaking is not in use and your receive buffer is too small, you run the risk of overflowing the buffer. As a general rule, start with the default size of 4096. If you experience errors, increase the buffer size to suit your application.

You cannot change the size of the input buffer once the port is open.

# Input Property

The Input property returns and removes a string of characters from the input buffer.

### Syntax

**[***string$* **=** **][***form***.]***Comm1***.Input**

### Data Type

String

### Usage

Read only at runtime only.

### Comments

The number of characters read from the buffer by Input at one time is determined by the InputLen property. Setting InputLen to zero tells Input to read the entire contents of the input buffer.

If the number of milliseconds specified in the InTimeout property passes before the number of characters specified in the InputLen property are read from the buffer using the Input property, PDQComm sets the CommEvent property to PDQ_ER_INTO, triggering the OnComm event.

### Example

This example shows how to poll for received data:

```
'--- specify to receive all available data
Comm1.InputLen = 0
Do
   '--- Allow other processes to continue
   Dummy = DoEvents()
   '--- Check for data
   If Comm1.InBufferCount Then
      '--- Read data
      In$ = PDQComm1.Input
   End If
Loop
```

# InputLen Property

The InputLen property sets and returns the number of bytes that will be read from the input buffer when the Input property is used.

### Syntax

**[***form.***]***Comm1*.InputLen**[** = *integer%***]**

### Data Type

Integer

### Usage

Read/write at design time and runtime.

### Comments

Use this property to read data from a machine whose output is formatted in fixed-length blocks of data.

If the number of milliseconds specified in the InTimeout property passes before the number of characters specified in the InputLen property are read from the buffer using the Input property, PDQComm sets the CommEvent property to PDQ_ER_INTO, triggering the OnComm event.

Setting InputLen to zero tells PDQComm to read the entire contents of the input buffer the next time Input is used.

### Example

This example shows how to read 10 bytes of data:

```
'--- Specify a 10 character frame of data
Comm1.InputLen = 10
'--- Read 10 bytes
TenBytes$ = Comm1.Input
```

# Interval Property

The Interval property sets the interval, in milliseconds, at which the comm control checks for events.

**Syntax**

**[***form*.**]***Comm1*.Interval**[** = *long&***]**

**Data Type**

Long Integer

**Usage**

Read/write at design time and runtime.

# InTimeout Property

The InTimeout property sets and returns the number of milliseconds to wait before signaling a failure when using the Input and LineInput properties to read data from the input buffer.

### Syntax

**[***form.***]***Comm1*.InTimeout**[** = *long&***]**

### Data Type

Long Integer

### Usage

Read/write at design time and runtime.

### Comments

If the number of milliseconds specified in the InTimeout property passes before the number of characters specified in the InputLen property are read from the buffer using the Input property, PDQComm sets the CommEvent property to PDQ_ER_INTO, triggering the OnComm event.

# KeyTranslation Property

The KeyTranslation property determines whether PDQComm translates certain non-character keys and sends them out of the serial port.

### Syntax

**[***form.***]***Comm1*.KeyTranslation**[** = *integer%***]**

### Data Type

Integer

### Usage

Read/write at design time and runtime.

### Comments

The KeyTranslation property only has an effect when AutoProcess is set to either PDQ_AUTOPROCESS_KEY or PDQ_AUTOPROCESS_BOTH. Valid settings are:

| Constant | Value | Description |
|---|---|---|
| PDQ_KEY_NONE | 0 | Default. No Translation Done. |
| PDQ_KEY_MANUAL | 1 | Not currently implemented |
| PDQ_KEY_ANSI | 2 | ANSI key translation |
| PDQ_KEY_VT52 | 3 | VT52 key translation |
| PDQ_KEY_VT100 | 4 | VT100 key translation |

When KeyTranslation is set to PDQ_KEY_VT100, the comm control translates the following keys to the apropriate VT100 escape sequences.

| | |
|---|---|
| CursorUp | (ESC) [A |
| CursorDown | (ESC) [B |
| CursorRight | (ESC) [C |
| CursorLeft | (ESC) [D |
| F1 | (ESC) OP |
| F2 | (ESC) OQ |
| F3 | (ESC) OR |
| F4 | (ESC) OS |

# LineInput Property

The LineInput property returns a complete line of text from the receive buffer. A line of text is defined as any string that ends with a Chr$ (13) carriage return.

### Syntax

**[***string$* **= ][***form***.]***Comm1***.LineInput**

### Data Type

String

### Usage

Read only at runtime only.

### Comments

Use LineInput to read strings of data that always end with a carriage return. If the number of milliseconds specified by InTimeout pass and there is no carriage return in the receive buffer, PDQComm sets the CommEvent property to PDQ_ER_INTO and fires the OnComm event. Set the InTimeOut property to a reasonable value so that the LineInput property can function properly.

If LineInput sees a carriage return, it deletes the carriage return and returns a string without a carriage return. It also deletes line feed characters as well. The result of LineInput is always a string without carriage returns or line feeds.

# NullDiscard Property

NullDiscard determines whether or not Chr$(0) NULL characters are transferred from the hardware port to the receive buffer.

### Syntax

**[***form***.]***Comm1*.NullDiscard**[** = *boolean%***]**

### Data Type

Boolean Integer

### Usage

Read/write at design time and runtime.

### Comments

Set NullDiscard to True to prohibit null characters from entering the receive buffer.

#  OutBufferCount Property

The OutBufferCount property returns the number of characters waiting in the output buffer. You can also use this property to to flush the output buffer.

### Syntax

**[***integer%* **= ][***form***.]***Comm1***.OutBufferCount**

### Data Type

Integer

### Usage

Read/write at runtime only.

### Comments

OutBufferCount refers to the number of characters that are waiting to be sent in the output buffer. The default output buffer size is 4096. You can flush the output buffer by setting the OutBufferCount property to zero.

Do not confuse this property with OutBufferSize, which reflects the total size of the output buffer.

# OutBufferSize Property

The OutBufferSize property sets and returns the size in bytes of the output buffer.

**Syntax**

**[***form*.**]***Comm1*.OutBufferSize**[** = *integer%***]**

**Data Type**

Integer

**Usage**

Read/write at design time and runtime.

**Comments**

Do not confuse this property with OutBufferCount, which reflects the number of bytes currently waiting in the output buffer. OutBufferSize refers to the total size of the output buffer. The default size is 4096 bytes.

You can not change the size of the output buffer once the port is open.

# Output Property

The Output property writes a string of characters to the output buffer.

**Syntax**

**[***form*.**]***Comm1*.Output**[** = *string$***]**

**Data Type**

String

**Usage**

Write only at runtime only.

**Comments**

The following example shows how to dial a phone number by sending a string to the modem through the serial port:

```
DialString$ = "ATDT555-1212" + Chr$(13)
Comm1.Output = DialString$
```

# ParityReplace Property

The ParityReplace property sets and returns the character that replaces an invalid character in the data stream when a parity error occurs.

**Syntax**

**[***form.***]***Comm1*.ParityReplace**[** = *string$***]**

**Data Type**

String

**Usage**

Read/write at design time and runtime.

**Comments**

The parity bit refers to a bit that is transmitted along with a specified number of data bits to provide a limited error checking capability. When you use a parity bit, PDQComm adds up all the bits that are set (having a value of 1) in the data and tests the sum as being odd or even according to the parity setting used when the port was opened.

If PDQComm detects a parity error, it discards the character and substitutes the character you specify to indicate the error. PDQComm also sets the CommEvent property to PDQ_ER_RXPARITY and fires the OnComm event.

Set ParityReplace to an empty string () to disable parity checking.

**NOTE** If you are connecting to the CompuServe Network, make sure ParityReplace is set to an empty string ().

# PortOpen Property

The PortOpen property sets and returns the state of the communications port (open or closed).

### Syntax

**[***form***.]***Comm1***.PortOpen[** = *boolean%***]**

### Data Type

Boolean Integer

### Usage

Read/write at runtime only.

### Comments

The CommPort property must be set to a valid port number before opening the port. To open the port, set the PortOpen property to True. Also, you must make sure that you specify a Baud rate in the Settings property that is supported by your modem and Windows communications driver. If the Settings property contains communications settings that your driver or modem does not support, PDQComm cannot access the port.

To close the port, set PortOpen to False. When you close the port, PDQComm automatically flushes the buffers. Use the following code to wait for any remaining data to be sent before closing the port:

```
Do
Dummy = DoEvents()                    'Enable control to finish sending
Loop Until Comm1.OutBufferCount = 0 'data until the buffer is
                                      ' empty
Comm1.PortOpen = False                'Then close the port.
```

PDQComm automatically closes the communications port when your application terminates.

### Example

The following example opens a communications port at 9600 baud, with no parity, with 8 data bits and 1 stop bit, and traps error #68 caused by opening an unavailable communications port:

```
On Error Resume Next
Comm1.Settings = "9600,N,8,1
"Do
   In$ = InputBox$("Enter comm Port:")
   PortNum = Val(In$)
   If PortNum Then
      Comm1.CommPort = PortNum
      Comm1.PortOpen = True
      If Err Then
         MsgBox "Port is invalid, try again
      "End If
   Else
      Exit Do
   End If
Loop
```

# Rows Property

The Rows property indicates the number of rows in the terminal window.

**Syntax**

**[***form***.]***Comm1***.Rows[** *= integer%***]**

**Data Type**

Integer

**Usage**

Read/write at design time and runtime.

**Comments**

The Rows property does not refer to the height of the control. Rather, it refers to the number of fixed-pitch rows when using terminal emulation, just as a standard DOS text screen has 25 rows.

**NOTE** If Emulation is set to PDQ_EMULATION_NONE (iconic), the Rows property is ignored.

# RThreshold Property

The RThreshold property sets and returns the number of characters to receive before PDQComm sets the CommEvent property to PDQ_EV_RECEIVE and triggers the OnComm event.

**Syntax**

**[***form.***]***Comm1*.RThreshold**[** = *integer%***]**

**Data Type**

Integer

**Usage**

Read/write at design time and runtime.

**Comments**

When you set RThreshold to 1, PDQComm fires the OnComm event whenever there is any data in the receive buffer.

When you set RThreshold to 0, PDQComm does not fire the OnComm event when characters are received. RThreshold has no effect when AutoProcess is in use.

**NOTE** Set RThreshold to 1 unless you are certain that you will only receive blocks of data that have a fixed length. Other values for the RThreshold property may cause PDQComm to lose characters. For example, if you set RThreshold to 2 and you receive a string of data with an odd number of bytes, PDQComm does not trigger the OnComm event when the last character is received because it is only one byte.

# RTSEnable Property

The RTSEnable property toggles the state of the RTS line.

**Syntax**

**[***form***.]***Comm1***.**RTSEnable**[** = *boolean%***]**

**Data Type**

Boolean Integer

**Usage**

Read/write at design time and runtime.

**Comments**

When RTSEnable is set to True the Request To Send line is set high (True) when the port is open, and low (False) when the port is closed. When RTSEnable is set to False, the RTS line is always low (unless RTS handshaking is used).

When you use RTS/CTS handshaking you must set the RTSEnable property to True, since the RTS line alternates between a high and low state when RTS/CTS handshaking is in use.

See the Handshaking property description for more information on handshaking protocols.

# ScrollText Property

The ScrollText property holds the contents of the scroll-back buffer.

### Syntax

**[***string$ =* **][***form.***]***Comm1*.ScrollText

### Data Type

String

### Usage

Read only at runtime only.

### Comments

The ScrollText property contains only the text of the data in the scroll-back buffer and not the screen codes. The scroll-back buffer is starts with the first line of text received after the ScrollRows property was set, and ends with the line before the top line of the terminal window.

# ScrollRows Property

The ScrollRows property sets and returns the number of rows of text to allocate to the scroll-back buffer.

**Syntax**

**[***form*.**]***Comm1*.ScrollRows**[** = *integer%***]**

**Data Type**

Integer

**Usage**

Read/Write at design time and runtime.

**Comments**

When you use terminal emulation, you can optionally allocate memory for a scroll-back buffer with the ScrollRows property. The scroll-back buffer holds text that has been received, allowing the user to scroll back to view previously received text.

See the topic Accessing the Scroll-Back Buffer for more information about the scroll-back buffer.

# Settings Property

The Settings property sets and returns the baud rate, parity, data bits and stop bits parameters.

**Syntax**

[*form.*]*Comm1*.Settings[ = *string$*]

**Data Type**

String

**Usage**

Read/Write at design time and runtime.

**Comments**

The setting string is composed of four settings and has the following format:

"*BBBB*,*P*,*D*,*S*"

Where *BBBB* is the baud rate, *P* is the parity, *D* is the number of data bits, and *S* is the number of stop bits.

Valid baud rates are determined by both the installed Windows communications driver, and the type of UART that your serial port uses.

**NOTE** Some PS/2 computers have UARTs that do not support baud rates greater than 19,200. Also, the Windows 3.0 API does not allow baud rates greater than 19,200.

The parity setting determines valid data bit and stop bit settings. The following table lists parity settings with corresponding valid data bit and stop bit values.

| Parity Setting | Description | Valid Data Bit Values | Valid Stop Bit Values |
|---|---|---|---|
| E | Even | 4 | 1 |
| N | None | 5 | 1.5 |
| O | Odd | 6 | 2 |
| S | Space | 7 | |
| M | Mark | 8 | |

**NOTE** If the Settings string is not valid when the port is opened, PDQComm display a Visual Basic Invalid Property Value dialog box. You can suppress the dialog box and handle the error manually by adding the line On Error Resume Next to your code, and testing the ERR function after setting the Settings property.

**Example**

The following example sets the control's port to communicate at 9600 Baud with no parity checking, 8 data bits and 1 stop bit:

```
Comm1.Settings = "9600,N,8,1"
```

# SmoothScroll Property

The SmoothScroll property determines whether the smooth scrolling feature is used.

**Syntax**

**[***form***.]***Comm1***.**SmoothScroll**[** **=** *boolean%***]**

**Data Type**

Boolean Integer

**Usage**

Read/write at design time and runtime.

**Comments**

When SmoothScroll is set to True, the text in the emulator window scrolls a pixel at a time as oposed to a line at a time. This produces a smooth-scrolling effect while sacrificing speed.

**NOTE** If you use smooth scrolling, use handshaking. Smooth scrolling slows the rate at which characters are read from the receive buffer. Under these conditions, the receive buffer may fill up and then you may start to lose characters.

# SThreshold Property

The SThreshold property sets and returns the minimum number of bytes allowable in the output buffer before PDQComm sets the CommEvent property to PDQ_EV_SEND and fires the OnComm event.

### Syntax

**[***form.***]***Comm1*.SThreshold**[** = *integer%***]**

### Data Type

Integer

### Usage

Read/write at design time and runtime.

### Comments

If the number of bytes in the output buffer is greater than SThreshold, PDQComm sets the CommEvent property is to PDQ_EV_SEND and fires the OnComm event.

Set the SThreshold property to 0 to disable the OnComm event for data transmission events.

# Text Property

The Text property sets and returns the text in the terminal window.

### Syntax

**[***form.***]***Comm1*.Text**[** = *filename$***]**

### Data Type

String

### Usage

Read/Write at runtime only.

### Comments

The Text property contains only the text of the data in the terminal window and not the screen codes. The text returned corresponds to the number of columns specified by the Columns property multiplied by the number of rows specified by the Rows property. This does not necessarily correspond to the portion of the terminal window that is currently visible on the display. PDQComm inserts a carriage return Chr$(13) at the end of each line of text.

When the Text property is assigned, PDQComm clears the terminal screen and displays the assigned text using the current terminal emulation and font settings.

**NOTE** The Text property does not contain the text in the scroll-back buffer. The ScrollText property contains all the text in the scroll-back buffer, and does not include the text in the terminal window.

See the Disp property for information about displaying text at the current cursor location.

### Example

This example clears the screen using the Text property and displays text using the Disp property:

```
'-- Save the current screen
Save$ = Comm1.Text
'-- Clear the terminal screen
Comm1.Text = ""
'-- Set the cursor to the top left
Comm1.CursorColumn = 1
Comm1.CursorRow = 1
'-- Display a string of text
Comm1.Disp = Text$
```

# Upload Property

The Upload property initiates a binary file transfer to upload (send) a file.

**Syntax**

**[***form***.]***Comm1***.Upload[** *= filename$***]**

**Data Type**

String

**Usage**

Read/Write at runtime only.

**Comments**

When the Upload property is set to a string that contains a valid filename, PDQComm starts an upload using the file transfer protocol defined by the XferProtocol property. Immediately after setting the Upload property, PDQComm returns control to your program. The transfer happens in the background and you can monitor the status of the transfer with the XferStatus property.

Use the built in file transfer dialog box to display the file transfer status. You may select the type of dialog box you want to display with the XferStatusDialog property.

You can set the Upload property several times to upload more than one file, no matter what file transfer protocol you use.

**NOTE** When XferStatus is greater than 0(file is being transferred), you cannot access the CommPort, Input, Output, LineInput, Settings or PortOpen properties, or set the InBufferCount and OutBufferCount properties.

See the topic File Transfers for more information on binary file transfers.

Also see the Upload subroutine in CTRLDEMO.BAS, which is used in CTRLDEMO.MAK, the PDQComm for Windows custom control demo terminal program.

**Examples**

The following example start an XModem upload using the built-in file transfer status dialog box:

```
'-- Set the protocol to XModem-Checksum
Comm1.XFerProtocol = PDQ_XMODEM_CHECKSUM
'-- Use the built-in dialog box to display status information
Comm1.XferStatusDialog = PDQ_XFERDIALOG_MODELESS
'-- Start the upload
Comm1.Upload = "FILE1.ZIP"
'-- Control immediately comes back here after setting the Upload
'property, and the file transfer will happen automatically in
'the background.
```

The following code start a YModem-Batch upload using the built-in file transfer status dialog box:

```
-- Set the protocol to YModem-Batch
Comm1.XFerProtocol = PDQ_YMODEM_BATCH
'-- Use the built-in dialog box to display status information
```

```
Comm1.XferStatusDialog = PDQ_XFERDIALOG_MODELESS
'-- Upload two files
Comm1.Upload = "FILE1.ZIP"
Comm1.Upload = "FILE2.ZIP"
'-- Control immediately comes back here after setting the Upload
'property, and the file transfers will happen
' automatically in the background.
```

Call DoEvents if you enter any loops after starting a file transfer to allow the system time to perform the transfer.

# XferCarrierAbort Property

The XferCarrierAbort property toggles automatic carrier detection durring file transfers.

**Syntax**

**[***form*.**]***Comm1*.XferCarrierAbort**[** = *boolean%***]**

**Data Type**

Boolean Integer

**Usage**

Read/write at design time and runtime.

**Comments**

When XferCarrierAbort is set to True, a file transfer requires the presence of CD in order to proceed. If you loose carrier in the middle of a file transfer, the transfer shuts down and returns control of the port back to your application. If you are using a direct connection, or something that doesn't assert the CD line, you must set this property to False in order to transfer files.

# XferDestFilename Property

The XferDestFilename property sets and returns the name of a file being transferred.

**Syntax**

**[***form***.]***Comm1*.XferDestFilename**[** = *filename$***]**

**Data Type**

String

**Usage**

Read/write at runtime only.

**Comments**

XferDestFilename always returns the name of the file to be written on the destination computer. For example, let's say we want to download a file from a BBS. On the BBS, the file is named BBS.ZIP.

However, we want this file to be written to our computer as, MYFILE.ZIP. To do this, we would simply set the Download property to the filename that we want to be written to disk:

```
Comm1.Download = "MYFILE.ZIP"
```

In this case, assuming that we are using a batch protocol, such as YModem-Batch, the XferSourceFilename is BBS.ZIP, and the XferDestFilename is MYFILE.ZIP The XferDestFilename property can only be set in the OnComm event after a PDQ_XFER event occurs, when the XferStatus property is either PDQ_XFER_FILE_READY or PDQ_XFER_FILE_START. The following sample code shows how to set the XferDestFilename property:

```
Sub Comm1_OnComm ()
   Select Case Comm1.CommEvent
   '--Did an XFER event occur?
   Case PDQ_XFER
      Select Case Comm1.XferStatus
         '--Is PDQComm preparing to transfer the file?
         Case PDQ_XFER_FILE_READY, PDQ_XFER_FILE_START
            '--Rename the file
            Comm1.XferDestFileName = "NEWNAME.ZIP"
      End Select
   End Select
End Sub
```

**NOTE** During an upload, there is no way to be certain that XferDestFilename is accurate, because the receiver has the ability to change the name of the file as it is being received.

See the XferSourceFilename property and the topic File Transfers for more information on binary file transfers.

# XferDialogHeight Property

The XferDialogHeight property controls the height in ScaleMode units of the file transfer dialog.

**Syntax**

*fDialHeight!* = **[***form***.]***Comm1*.XferDialogHeight

**Data Type**

Single Precision

**Usage**

Read only at design time and runtime.

**Comments**

Use this property with XferDialogWidth to calculate positions on the screen that center the dialog. For example, the following code centers the file transfer dialog on the screen:

```
Comm1.XferDialogLeft = (Screen.Width - Comm1.XferDialogWidth) \ 2
Comm1.XferDialogTop = (Screen.Height - Comm1.XferDialogHeight) \ 2
```

# XferDialogLeft Property

The XferDialogLeft property determines the position of the left edge of the file transfer dialog box.

### Syntax

**[***form*.**]***Comm1*.XferDialogLeft **[** = *single!* **]**

### Data Type

Single Precision

### Usage

Read/Write at design time and runtime.

### Comments

This property is in absolute screen position in ScaleMode units. By default, this property is set to -1, which positions the top left corner of the dialog over the top left corner of the PDQComm control.

# XferDialogTop Property

The XferDialogTop property determines the position of the top edge of the file transfer dialog box.

### Syntax

**[***form*.**]***Comm1*.XferDialogTop **[** = *single!* **]**

### Data Type

Single Precision

### Usage

Read/Write at design time and runtime.

### Comments

This property is in absolute screen position in ScaleMode units. By default, this property is set to -1, which positions the top left corner of the dialog over the top left corner of the PDQComm control.

# XferDialogWidth Property

The XferDialogWidth property contains the width in ScaleMode units of the file transfer dialog.

### Syntax

*fDialWidth!* = **[***form***.]***Comm1*.XferDialogWidth

### Data Type

Single Precision

### Usage

Read only at design time and runtime.

### Comments

Use this property with XferDialogHeight to calculate positions on the screen that center the dialog. See the XferDialogHeight property for more information.

# XferFileDate Property

The XferFileDate property returns the date of the last modification for file being transferred.

**Syntax**

**[***string&* **= ][***form***.]***Comm1***.XferFileDate**

**Data Type**

String

**Usage**

Read only at runtime only.

**Comments**

See the topic File Transfers for more information on binary file transfers.

# XferFileSize Property

The XferFileSize property returns the size (in bytes) of a file being transferred.

**Syntax**

**[***long&* **=** **][***form***.]***Comm1***.XferFileSize**

**Data Type**

Long Integer

**Usage**

Read only at runtime only.

**Comments**

See the topic File Transfers for more information on binary file transfers.

# XferFileTime Property

The XferFileTime property returns the time of the last modification for a file being transferred.

**Syntax**

**[***string&* **= ][***form***.]***Comm1***.XferFileTime**

**Data Type**

String

**Usage**

Read only at runtime only.

**Comments**

See the topic File Transfers for more information on binary file transfers.

# XferMessage Property

The XferMessage property returns a status message from a file transfer in progress.

**Syntax**

[*string$ =* ][*form.*]*Comm1*.XferMessage

**Data Type**

String

**Usage**

Read only at runtime only.

**Comments**

XferMessage returns the status of a file transfer in progress in the form of a string.

You can use this property to display information in your own custom file transfer status dialog box.

See the topic File Transfers for more information on binary file transfers.

# XferProtocol Property

The XferProtocol property sets and returns the transfer protocol for a file transfer.

**Syntax**

`[form.]Comm1.XferProtocol[ = integer%]`

**Data Type**

Integer

**Usage**

Read/Write at design time and runtime.

**Comments**

The following file transfer protocol constants are defined in COMMCTRL.BAS, which should be included in your project if you are using the PDQComm for Windows custom control:

Valid settings are:

| Constant | Value | Protocol |
|---|---|---|
| PDQ_XMODEM_CHECKSUM | 0 | XModem-Checksum |
| PDQ_XMODEM_CRC | 1 | XModem-CRC |
| PDQ_XMODEM_1K | 2 | XModem-1K |
| PDQ_YMODEM_BATCH | 3 | YModem-Batch |
| PDQ_YMODEM_G | 4 | YModem-G |
| PDQ_ZMODEM | 5 | ZModem |
| PDQ_COMPUSERVE_BPLUS | 6 | CompuServe B+ |
| PDQ_KERMIT | 7 | Kermit |

**NOTE** For the current list of supported protocols, see the README.WRI file in the PDQComm product directory.

See the topic File Transfers for more information on binary file transfers.

# XferSourceFilename Property

The XferSourceFilename property returns the original name of a file during a file transfer.

### Syntax

**[***form***.]***Comm1*.XferSourceFilename**[** = *filename$***]**

### Data Type

String$

### Usage

Read only at runtime only.

### Comments

During an upload, the XferSourceFilename property returns the original filename, or the name of the file on your computer's disk.

During a download, the XferSourceFilename property returns the original name of the file on the other computer's disk, but only if the protocol being used is a batch protocol, such as YModem-Batch or ZModem. If you are using a non-batch protocol, such as XModem, the XferSourceFilename property will return an empty string ().

See the XferDestFilename property description and the topic File Transfers for more information on binary file transfers.

# XferStatus Property

The XferStatus property sets and returns the status of a file transfer that is in progress.

**Syntax**

`[integer% = ][form.]Comm1.XferStatus`

**Data Type**

Integer

**Usage**

Read/Write at runtime only.

**Comments**

Use this property inside the communications event handler when CommEvent is set to PDQ_EV_XFER. You can also use it to determine the behavior of the file transfer or to display information in your own custom file transfer status dialog box.

See the topic File Transfers for more information on binary file transfers.

| Constant | Value | Description |
|---|---|---|
| PDQ_XFER_TERM_OK | 0 | The entire transfer completed successfully. |
| PDQ_XFER_WAITING | 1 | PDQComm is waiting for the other computer. |
| PDQ_XFER_FILE_READY | 2 | PDQComm is preparing to transfer a file. The XferSourceFilename and XferDestFilename properties will contain the name of the file to be transferred. When you get this code, you can change the name of the file to transfer by assigning a value to the XferDestFilename property. If you wish to abort the transfer you can assign XferStatus property to 5 (skip file) or 6 (abort entire transfer) |
| PDQ_XFER_FILE_START | 3 | PDQComm is starting to transfer a file. At this point the XferDestFilename property cannot be changed, and the XferFileSize is filled if available. |
| PDQ_XFER_XFERING | 4 | PDQComm is transferring data. This event is generated when XferMessage or XferTransferred have changed. |
| PDQ_XFER_SKIP | 5 | PDQComm is skipping the next file. This can also be assigned to abort or skip transferring the current file when XferStatus is PDQ_XFER_FILE_READY |
| PDQ_XFER_ABORT | 6 | PDQComm is aborting the current file transfer. This can also be assigned to abort or skip the transferring of the current file when XferStatus is PDQ_XFER_FILE_READY, PDQ_XFER_FILE_START, or PDQ_XFER_XFERING. |
| PDQ_XFER_FINISHED | 7 | PDQComm finished transferring the current file. XferDestFilename contains the filename |

| | | |
|---|---|---|
| | | that was transferred. The file is closed so you can copy it, rename it, or do whatever you want with it. |
| PDQ_XFER_LOSTCARRIER | 8 | When the XferCarrierAbort property is set to True, Carrier Detect monitoring is enabled. A loss of carrier will cause the transfer to abort and fire this event. |
| PDQ_XFER_TIMEOUT | 9 | On an upload, this event gets fired every 10 seconds while waiting on the remote to start the download. You have the option of aborting the transfer by setting XferStatus to PDQ_XFER_ABORT |
| PDQ_XFER_TERM_ERROR | -1 | The entire transfer did not completed successfully. |

# XferStatusDialog Property

The XferStatusDialog property sets the type of file transfer status dialog box that will be used for file transfers.

**Syntax**

**[***form***.]***Comm1*.XferStatusDialog**[** = *integer%***]**

**Data Type**

Integer

**Usage**

Read/Write at design time and runtime.

**Comments**

| Constant | Value | Description |
|---|---|---|
| PDQ_XFERDIALOG_NONE | 0 | No dialog box will be shown. This is useful if you want to create your own file transfer dialog box. |
| PDQ_XFERDIALOG_MODELESS | 1 | The dialog box will be modeless (or not-modal). |
| PDQ_XFERDIALOG_MODAL | 2 | The dialog box will be modal |

See the topic File Transfers for more information on binary file transfers.

# XferTransferred Property

The XferTransferred property returns the number of bytes that have been transferred in a file transfer that is in progress.

**Syntax**

**[***long&* **=** **][***form***.]***Comm1***.XferTransferred**

**Data Type**

Long Integer

**Usage**

Read only at runtime only.

**Comments**

You can use this property to display information in your own custom file transfer status dialog box.

See the topic File Transfers for more information on binary file transfers.

# CloseCommHandle Method

Closes a comm port that was previously opened with OpenCommHandle.

### Syntax

```
bSuccess = [form.]Comm1.CloseCommHandle
```

### Parameters

None.

### Returns

TRUE if successful, FALSE if unsuccessful.

### Comments

This method deactivates a PDQComm control's control of the port opened with the OpenCommHandle method. The CloseCommHandle method does not close the comm port. Use this method to manage access by several controls/applications to a single a port.

# CRC16 Method

Returns the CRC16 value of a block of data.

### Syntax

```
MyCRC16 = [form.]Comm1.CRC16(varData$)
```

### Parameters

*varData* - The data for which you want to calculate a CRC16 value.

### Returns

A two byte string that represents the CRC16 value of the data.

### Comments

Since this parameter is a variant, you can pass it any data type including a control. However, due to the way variants are coerced, passing an integer causes PDQComm to calculate the CRC value based on the ASCII representation of the integer instead of the binary representation. Therefore, pass only String data to this method.

# CRC32 Method

Returns the CRC32 value of a block of data.

### Syntax

```
MyCRC32 = [form.]Comm1.CRC32(varData$)
```

### Parameters

*varData* - The data for which you want to calculate a CRC32 value.

### Returns

A four byte string that represents the CRC32 value of the data.

### Comments

Since this parameter is a variant, you can pass it any data type including a control. However, due to the way variants are coerced, passing an integer causes PDQComm to calculate the CRC value based on the ASCII representation of the integer instead of the binary representation. Therefore, pass only String data to this method.

# OpenCommHandle Method

Enables a PDQComm control to take control of an open comm port using that ports handle.

**Syntax**

```
bSuccess = [form.]Comm1.OpenCommHandle(hComm&)
```

**Parameters**

*hComm* - A handle to the open comm port. For PDQComm, this is the value of the CommID property.

**Returns**

TRUE if successful, FALSE if unsuccessful.

**Comments**

This method allows a PDQComm control to take control of an open comm port. Remember to call CloseCommHandle when you finish using the port. Use this method to manage access by several controls/applications to a single a port.

**NOTE** Do not use the PortOpen property when using these methods.

# SuspendComm Method

Allows a PDQComm control to suspend or resume processing on an open comm port.

### Syntax

```
bSuccess = [form.]Comm1.SuspendComm(bAction%)
```

### Parameters

*bAction* - A boolean value that determines whether the method suspends or resumes processing for a port. A value of TRUE suspends processing on an open port, while a value of FALSE resumes processing.

### Returns

TRUE if successful, FALSE if unsuccessful.

### Comments

This method allows a PDQComm control to suspend processing on an open comm port. This allows you to pass the comm port handle to another PDQComm control or application. When you are ready to take back control, you call this method with *bAction* parameter set to FALSE. Use this method to manage access by several controls/applications to a single a port.

# OnComm Event

The OnComm event occurs whenever the value of the CommEvent property changes, indicating that either a communications event or error occurred.

### Syntax

```
Subname_OnComm()
```

### Comments

The actual error or event that triggered the OnComm event is reflected in the CommEvent property.

**NOTE** Note that setting the RThreshold or SThreshold properties to 0 disables trapping for the PDQ_EV_RECEIVE and PDQ_EV_SEND events respectively.

### Example

This example shows how to handle communications errors and events. You can insert code to handle a particular error or event after its corresponding Case statement below.

```
Sub Comm1_OnComm
  Select Case Comm1.CommEvent
   '-- Errors
   Case PDQ_ER_BREAK    'A Break was received.
   '--- Code to handle a BREAK goes here, etc
   Case PDQ_ER_INTO        'Input Timeout
   Case PDQ_ER_CDTO        'CD (RLSD) Timeout.
   Case PDQ_ER_CTSTO    'CTS Timeout.
   Case PDQ_ER_DSRTO    'DSR Timeout.
   Case PDQ_ER_FRAME    'Framing Error
   Case PDQ_ER_OVERRUN     'Data Lost.
   Case PDQ_ER_RXOVER      'Input buffer overflow.
   Case PDQ_ER_RXPARITY 'Parity Error.
   Case PDQ_ER_TXFULL      'Output buffer full.

   '-- Events
   Case PDQ_EV_CD       'Change in the CD line.
   Case PDQ_EV_CTS         'Change in the CTS line.
   Case PDQ_EV_DSR         'Change in the DSR line.
   Case PDQ_EV_RING        'Change in the Ring Indicator.
   Case PDQ_EV_RECEIVE     'Received RThreshold number of characters.
   Case PDQ_EV_SEND        'There are (SThreshold - 1)
                        'number of characters in the output buffer.
   Case PDQ_EV_XFER        'File Transfer event. Check the XferStatus
                        'property to determine the transfer event.
  End Select
End Sub
```

# PDQComm Control Constants

Several properties of the PDQComm control take enumerated values. These properties can only accept values from a discrete list of integer values. Each integer value represents a behavior setting for the property. To make your code more readable, the PDQComm control supports the use of constants for reading and setting the value of an enumerated property. The PDQComm OCXs automatically establish constants for enumerated property values.

**AutoProcess**

| Constant | Value | Description |
|---|---|---|
| PDQ_AUTOPROCESS_NONE | 0 | Comm control does not automatically process serial or keyboard input. If AutoProcess is set to 0, the PDQ_EV_RECEIVE event will not fire if RThreshold is 0. See the RThreshold property and the OnComm event for more information. |
| PDQ_AUTOPROCESS_SERIAL | 1 | All data that is received by the serial port is automatically displayed in the terminal window (assuming that the Emulation property is non-zero). |
| PDQ_AUTOPROCESS_KEY | 2 | All keys pressed in the terminal window are sent out the serial port (assuming that the Emulation property is non-zero). |
| PDQ_AUTOPROCESS_BOTH | 3 | PDQComm acts as a terminal program, both displaying data that comes in over the serial port and also sending keystrokes out the serial port. |

**AutoScroll**

| Constant | Value | Description |
|---|---|---|
| PDQ_AUTOSCROLL_NONE | 0 | The terminal window does not scroll automatically. |
| PDQ_AUTOSCROLL_VERTICAL | 1 | The terminal window scrolls vertically when the cursor goes beyond the bottom-most row. |
| PDQ_AUTOSCROLL_HORIZONTAL | 2 | The terminal window scrolls horizontally when the cursor goes beyond the right-most column |
| PDQ_AUTOSCROLL_BOTH | 3 | The terminal window scrolls vertically when the cursor goes beyond the bottom-most row, and horizontally when the cursor goes beyond the right-most column |
| PDQ_AUTOSCROLL_VERTKEY | 4 | The terminal window scrolls vertically when the cursor goes beyond the bottom-most row, and only scrolls horizontally if you type beyond the right-most column. |

**Backspace**

| Constant | Value | Description |
|---|---|---|
| PDQ_BACKSPACE_DESTRUCTIVE | 0 | Backspace characters are destructive, and erase the previous character in the terminal window (default). |
| PDQ_BACKSPACE_NON_DESTRUCTIVE | 1 | Backspace characters are non-destructive, and do not erase the previous character in the terminal window. |

**Capture**

| Constant | Value | Description |
|---|---|---|
| PDQ_CAPTURE_STANDARD | 0 | Data is saved without color codes in the exact order that it was received. For example, you could receive an ANSI string to draw a line at row 20, followed by an ANSI string to draw a line at row 19. The capture file would contain row 20 followed by row 19, not the way it appears on the screen |
| PDQ_CAPTURE_BINARY | 1 | Nothing is filtered and data is saved verbatim. |
| PDQ_CAPTURE_VISIBLE | 2 | Only the text is saved and written to disk, exactly as it looks in the terminal window (without the color codes). Because of the way PDQ_CAPTURE_VISIBLE works, data is required to scroll off the screen, to be saved to the file. If the incoming data is sending control codes to erase the screen with scrolling, data will not be saved to the capture file. |

**ColorFilter**

| Constant | Value | Description |
|---|---|---|
| PDQ_COLOR_FULL | 0 | The BackColor and ForeColor are active in full color. |
| PDQ_COLOR_GREY | 1 | All color values are scaled to gray. |
| PDQ_COLOR_MONO | 2 | All color values are scaled to the BackColor and ForeColor that are currently set. You would use gray scaled or monochrome colors if, for example, you are running on a laptop with 64-shade VGA video hardware. |

**Cursor Type**

| Constant | Value | Description |
|---|---|---|
| PDQ_CURSOR_VBAR | 0 | Vertical bar cursor |
| PDQ_CURSOR_BLOCK | 1 | Block cursor |

**Emulation**

| Constant | Value | Description |
|---|---|---|
| PDQ_EMULATION_NONE | 0 | The control is invisible at run-time and iconic at design-time. Although the control is invisible at runtime, it will continue to process serial communications. |
| PDQ_EMULATION_TTY | 1 | Emulation window handles standard DOS control codes but does not support color or cursor position sequences. |
| PDQ_EMULATION_ANSI | 2 | Supports ANSI escape sequences allowing for color and cursor positioning. |
| PDQ_EMULATION_VT52 | 3 | Supports VT52 escape sequences allowing for color and cursor positioning. |
| PDQ_EMULATION_VT100 | 4 | Supports VT100 escape sequences allowing for color and cursor positioning. |

**Errors**

| Constant | Value | Description |
|---|---|---|
| PDQ_ER_BREAK | 1001 | A Break signal was received |
| PDQ_ER_CTSTO | 1002 | Clear To Send Timeout. The CTS line was low for CTSTimeout number of milliseconds while trying to transmit a character. The CTSTimeout property is obsolete. |
| PDQ_ER_DSRTO | 1003 | Data Set Ready Timeout. The DSR line was low for DSRTimeout number of milliseconds while trying to transmit a character. The DSRTimeout property is obsolete. |
| PDQ_ER_FRAME | 1004 | Framing Error. The hardware detected a framing error. Generally, you cannot do anything about this error, except to try to send or receive again. |
| PDQ_ER_INTO | 1005 | Input Timeout. The requested number of characters specified by the InputLen property could not be returned by the Input or LineInput properties before InTimeout number of milliseconds have passed |
| PDQ_ER_OVERRUN | 1006 | Port Overrun. A character was not read from the hardware before the next character arrived, and was thus lost. If you receive this error, try lowering your baud rate or upgrading your serial port hardware |
| PDQ_ER_CDTO | 1007 | Carrier Detect Timeout. The CD line |

| | | |
|---|---|---|
| | | was low for CDTimeout number of milliseconds while trying to transmit a character. Carrier Detect is also known as the Receive Line Signal Detect (RLSD). The CDTimeout property is obsolete. |
| PDQ_ER_RXOVER | 1008 | Receive Buffer Overflow. There is no room in the input buffer. |
| PDQ_ER_RXPARITY | 1009 | Parity Error. The hardware detected a parity error. |
| PDQ_ER_TXFULL | 1010 | Transmit Buffer Full. The output buffer is full while trying to queue a character. |

**Events**

| Constant | Value | Description |
|---|---|---|
| PDQ_EV_SEND | 1 | There are fewer than SThreshold number of characters in the output buffer |
| PDQ_EV_RECEIVE | 2 | Received RThreshold number of characters. |
| PDQ_EV_CTS | 3 | Change in Clear To Send line. |
| PDQ_EV_DSR | 4 | Change in Data Set Ready line. |
| PDQ_EV_CD | 5 | Change in Carrier Detect line. |
| PDQ_EV_RING | 6 | Ring detected. |
| PDQ_EV_EOF | 7 | End Of File (ASCII character 26) character received. |
| PDQ_EV_ZMODEM | 8 | ZModem startup sequence. |
| PDQ_EV_XFER | 100 | File transfer event. Read the XferStatus property for the current file transfer status. |

**Handshake**

| Constant | Value | Description |
|---|---|---|
| PDQ_HANDSHAKING_NONE | 0 | No handshaking is used. Beware of possible buffer overruns. |
| PDQ_HANDSHAKING_XON | 1 | Software handshaking is used. Control codes are sent between systems to control the flow of data. |
| PDQ_HANDSHAKING_RTS | 2 | Hardware handshaking is used. You should use RTS handshaking when communicating with a high speed modem. |
| PDQ_HANDSHAKING_BOTH | 3 | Hardware and software handshaking is used. |

**Key Translation**

| Constant | Value | Description |
|---|---|---|
| PDQ_KEY_NONE | 0 | Default. No Translation Done. |
| PDQ_KEY_MANUAL | 1 | Not currently implemented |

| | | |
|---|---|---|
| PDQ_KEY_ANSI | 2 | ANSI key translation |
| PDQ_KEY_VT52 | 3 | VT52 key translation |
| PDQ_KEY_VT100 | 4 | VT100 key translation |

**PDQColorConstants**

| Color | Value | RGB Value (Hex) |
|---|---|---|
| PDQ_COLOR_BLACK | 0 | &H000000 |
| PDQ_COLOR_BLUE | 1 | &H800000 |
| PDQ_COLOR_GREEN | 2 | &H008000 |
| PDQ_COLOR_CYAN | 3 | &H808000 |
| PDQ_COLOR_RED | 4 | &H000080 |
| PDQ_COLOR_MAGENTA | 5 | &H800080 |
| PDQ_COLOR_YELLOW | 6 | &H008080 |
| PDQ_COLOR_WHITE | 7 | &HC0C0C0 |
| PDQ_COLOR_GRAY | 8 | &H808080 |
| PDQ_COLOR_LIGHTBLUE | 9 | &HFF0000 |
| PDQ_COLOR_LIGHTGREEN | 10 | &H00FF00 |
| PDQ_COLOR_LIGHTCYAN | 11 | &HFFFF00 |
| PDQ_COLOR_LIGHTRED | 12 | &H0000FF |
| PDQ_COLOR_LIGHTMAGENTA | 13 | &HFF00FF |
| PDQ_COLOR_LIGHTYELLOW | 14 | &H00FFFF |
| PDQ_COLOR_BRIGHTWHITE | 15 | &HFFFFFF |

**XferProtocol**

| Constant | Value | Protocol |
|---|---|---|
| PDQ_XMODEM_CHECKSUM | 0 | XModem-Checksum |
| PDQ_XMODEM_CRC | 1 | XModem-CRC |
| PDQ_XMODEM_1K | 2 | XModem-1K |
| PDQ_YMODEM_BATCH | 3 | YModem-Batch |
| PDQ_YMODEM_G | 4 | YModem-G |
| PDQ_ZMODEM | 5 | ZModem |
| PDQ_COMPUSERVE_BPLUS | 6 | CompuServe B+ |
| PDQ_KERMIT | 7 | Kermit |

**XferStatus**

| Constant | Value | Description |
|---|---|---|
| PDQ_XFER_TERM_OK | 0 | The entire transfer completed successfully. |
| PDQ_XFER_WAITING | 1 | PDQComm is waiting for the other computer. |
| PDQ_XFER_FILE_READY | 2 | PDQComm is preparing to transfer a file. The XferSourceFilename and XferDestFilename properties will contain the name of the file to be transferred. When you get this code, you can change the name of the file to transfer by assigning a value to |

| | | |
|---|---|---|
| | | the XferDestFilename property. If you wish to abort the transfer you can assign XferStatus property to 5 (skip file) or 6 (abort entire transfer) |
| PDQ_XFER_FILE_START | 3 | PDQComm is starting to transfer a file. At this point the XferDestFilename property cannot be changed, and the XferFileSize is filled if available. |
| PDQ_XFER_XFERING | 4 | PDQComm is transferring data. This event is generated when XferMessage or XferTransferred have changed. |
| PDQ_XFER_SKIP | 5 | PDQComm is skipping the next file. This can also be assigned to abort or skip transferring the current file when XferStatus is PDQ_XFER_FILE_READY |
| PDQ_XFER_ABORT | 6 | PDQComm is aborting the current file transfer. This can also be assigned to abort or skip the transferring of the current file when XferStatus is PDQ_XFER_FILE_READY, PDQ_XFER_FILE_START, or PDQ_XFER_XFERING. |
| PDQ_XFER_FINISHED | 7 | PDQComm finished transferring the current file. XferDestFilename contains the filename that was transferred. The file is closed so you can copy it, rename it, or do whatever you want with it. |
| PDQ_XFER_LOSTCARRIER | 8 | When the XferCarrierAbort property is set to True, Carrier Detect monitoring is enabled. A loss of carrier will cause the transfer to abort and fire this event. |
| PDQ_XFER_TIMEOUT | 9 | On an upload, this event gets fired every 10 seconds while waiting on the remote to start the download. You have the option of aborting the transfer by setting XferStatus to PDQ_XFER_ABORT |
| PDQ_XFER_TERM_ERROR | -1 | The entire transfer did not complete successfully. |

**XferStatusDialog**

| Constant | Value | Description |
|---|---|---|
| PDQ_XFERDIALOG_NONE | 0 | No dialog box will be shown. This is useful if you want to create your own file transfer dialog box. |
| PDQ_XFERDIALOG_MODELESS | 1 | The dialog box will be modeless (or |

| | | |
|---|---|---|
| | | not-modal). |
| PDQ_XFERDIALOG_MODAL | 2 | The dialog box will be modal |

# PDQComm Emulation Properties

The following properties allow you to set the appearance and behavior of a terminal window and emulation for the current Comm control:

AnswerBack
Appearance
AutoScroll
AutoSize
BackColor
Backspace
CaptureMode
ColorFilter
Columns, Rows
CursorType
Echo
Emulation
ForeColor
KeyTranslation
ScrollRows
SmoothScroll

**Save As Default**

Save the current property settings for the PDQComm control to the configuration file (.INI) associated with the current application

**Reset Default**

Reset the property settings to the default PDQComm control settings.

**About**

Display information about the currently installed version of the PDQComm control.

# PDQComm Hardware (General) Properties

The following properties allow you to configure the communications port, line, and modem associated with the current Comm control:

| | |
|---|---|
| AutoProcess | Handshaking |
| CommPort | RTSEnable |
| DTREnable | Settings |

# PDQComm Hardware (Advanced) Properties

The following properties allow you to configure buffers and data-stream handling behavior for the current Comm control:

| | |
|---|---|
| InBufferSize | OutBufferSize |
| InputLen | ParityReplace |
| InTimeOut | RThreshold |
| NullDiscard | SThreshold |

# PDQComm File Transfer Properties

The following properties allow you to configure file transfer behavior for the current Comm control:

AutoZModem

XferCarrierAbort

XferProtocol

XferStatusDialog

# AsciiSend

**Visual Basic Subroutine Contained In MWXFER.BAS**

Sends (uploads) a text file.

**Syntax**

```
Call AsciiSend(Comm1, FileName$, CDFlag%)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |
| *FileName$* | Specifies the file to send. |
| *CDFlag* | When True tells AsciiSend to abort when carrier is lost. |

**Comments**

AsciiSend sets MWError () to one of the following values:

| **Value** | **Description** |
|---|---|
| MW_ER_CANCELED | Operation was Canceled. |
| MW_ER_LOST_CARRIER | Carrier was lost. The call was disconnected. |
| MW_ER_PORTCLOSED | The specified port does not exist. |
| MW_ER_CTRLERROR | The specified port can not be accessed. |
| MW_ASCIISEND_OK | File sent ok. |
| MW_ASCIISEND_NOTFOUND | File not found. |
| MW_ASCIISEND_FILE_ERR | Error reading file. |

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, MWXFER.BAS, MWXFER.FRM, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# Download

**Visual Basic Subroutine Contained In MWXFER.BAS**

Downloads (receives) a file, letting the user pick the transfer protocol and filename.

**Syntax**

```
Call Download(Comm1)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |

**Comments**

Download displays the protocol dialog box, which contains a list of all file transfer protocols supported by PDQComm. If the user picks a protocol that requires the specification of a file name, PDQComm displays the file dialog box. If the protocol does not require a file name (Ymodem-G, Ymodem-Batch, Zmodem, Compuserve B+) PDQComm does not display the file dialog box. The download occurs in the background, during which PDQ_EV_XFER events fire via the OnComm event. When one of these events occurs, the XferStatus property logs the status of the transfer.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, MWPROTCL.FRM, MWXFER.BAS, MWXFER.FRM, COMDLG32.OCX or COMDLG16.OCX, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# Hangup

**Visual Basic Subroutine Contained In MWMODEM.BAS**

Terminates the call.

**Syntax**

```
Call Hangup(Comm1, NumTries%, UseDTR%)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |
| *NumTries%* | Specifies the number of times to attempt a Hangup. |
| *UseDTR%* | Specifies whether the routine should drop DTR. |

**Comments**

Hangup sets MWError () to one of the following values:

| **Value** | **Description** |
|---|---|
| MW_ER_CANCELED | Operation was Canceled. |
| MW_ER_LOST_CARRIER | Carrier was lost. The call was disconnected. |
| MW_ER_PORTCLOSED | The specified port does not exist. |
| MW_ER_CTRLERROR | The specified port can not be accessed. |
| MW_ER_TIMEOUT | Timeout. |
| MW_HANGUP_OK | Modem was disconnected. |
| MW_HANGUP_ERROR | Modem could not be disconnected. |
| MW_HANGUP_OFFLINE | Modem is already hung up. |

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, MWMODEM.BAS, MWMODEM.FRM, MWMODSTR.FRM, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# LoadPortSettings

**Visual Basic Subroutine Contained In MWPORT.BAS**

Loads port settings from the configuration file, and opens the port.

**Syntax**

```
Call LoadPortSettings(Comm1)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |

**Comments**

LoadPortSettings reads the port settings from the configuration file. If the file or the settings cannot be found, LoadPortSettings calls SetPortSettings, which displays a dialog box containing settings for port, baud rate, parity, data bits, stop bits, duplex (echo), DTR, RTS, and handshaking. When the user presses the OK button in this dialog, PDQComm initializes the port and saves the port information to the configuration file.

The ModemWare setup routines store your port, terminal, and modem settings in a file in the \WINDOWS directory. The name of this file is derived from the name of your application, with an INI extension. For example, if this program is called PROJECT1.VBP then your configuration file name is PROJECT1.INI. This file will always reside in the \WINDOWS directory.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, MWPORT.BAS, MWPORT.FRM, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

**See Also**

SetPortSettings

# LoadTerminalSettings

**Visual Basic Subroutine Contained In MWTERM.BAS**

Loads the terminal settings from the configuration file.

**Syntax**

```
Call LoadTerminalSettings(Comm1)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |

**Comments**

LoadTerminalSettings reads the terminal emulator settings from the configuration file. If the file or settings cannot be found, LoadTerminalSettings calls SetTerminalSettings, which displays a dialog box containing settings for emulation, fonts, colors, and the scrollback buffer size. When the user presses the OK button in this dialog, PDQComm resets the terminal window and saves the settings to the configuration file.

The ModemWare setup routines store your port, terminal, and modem settings in a file in the \WINDOWS directory. The name of this file is derived from the name of your application, with an INI extension. For example, if this program is called PROJECT1.VBP, then your configuration file name is PROJECT1.INI. This file will always reside in the \WINDOWS directory.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, MWTERM.BAS, MWTERM.FRM, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

**See Also**

SetTerminalSettings

# LoadModemSettings

**Visual Basic Subroutine Contained In MWMODEM.BAS**

Loads the modem settings from the configuration file.

**Syntax**

```
Call LoadModemSettings(Comm1)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |

**Comments**

LoadModemSettings loads the initialization string and other modem command strings from the configuration file. If the file or the information cannot be found, LoadModemSettings calls SetModemSettings, which displays a dialog box containing a list of over 440 modems. When the user picks his or her modem from the list and clicks the Change button in this dialog, PDQComm initializes the modem and saves the information to the configuration file.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, MWMODEM.BAS, MWMODEM.FRM, MWMODSTR.FRM, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

**See Also**

SetModemSettings

# MW_CancelFlag

**Global Variable Contained In MODMWARE.BAS**

Cancels the current ModemWare process when set to True.

**Syntax**

```
MW_CancelFlag = {True|False}
```

**Comments**

Use this variable to cancel any ModemWare process, such as dialing a number.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# MW_CurRoutine

**Global Variable Contained In MODMWARE.BAS**

Returns a code which specifies a currently executing ModemWare routine.

**Syntax**

```
If MW_CurRoutine = MW_PROC_HANGUP Then
   MsgBox "Hanging Up"
End If
```

**Comments**

Use this variable to find out which routine is executing at any time. It is particularly useful debugging, or if you just want to know what code is doing. Since it is very difficult to debug communications code in real time, MW_CurRoutine provides a way to log what ModemWare routines are in use in a particular situation.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# MW_DisplayFlag

**Global Variable Contained In MODMWARE.BAS**

Displays or suppresses the display of incoming text for a WaitFor routines in the terminal window.

**Syntax**

```
MW_DisplayFlag =  {True|False}
```

**Comments**

Set MW_DisplayFlag to False if you do not wish the user to see what is being received by a WaitFor routine or if you are not using emulation. PDQComm does not save this value in the configuration file. The default value for this variable is True.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# MWEnd

**Visual Basic Subroutine Contained In MWMODEM.BAS**

Disconnects the modem (if it is online) and closes the comm port.

**Syntax**

```
Call MWEnd(Comm1)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, MWMODEM.BAS, MWMODEM.FRM, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# MWError

**Visual Basic Function Contained In MODMWARE.BAS**

Returns the current ModemWare error.

**Syntax**

```
ErrorCode% = MWError()
```

Where:

| | |
|---|---|
| *ErrorCode%* | Returns the current error code. |

**Data Type**

Integer

**Comments**

You should check MWError after calls to any ModemWare routine that sets the ModemWare error. For information about the error code returned by a ModemWare routine, see the reference entry for the routine in question.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# MWErrorMsg

**Visual Basic Function Contained In MODMWARE.BAS**

Returns a string describing the current ModemWare Error.

**Syntax**

```
Msg$ = MWErrorMsg$()
```

Where:

| | |
|---|---|
| *Msg$* | Returns a string describing the last error. |

**Data Type**

String

**Comments**

Use the MWErrorMsg$ function to return a string to display on a status bar to provide the user information about the current state of the application.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# MWPause

**Visual Basic Subroutine Contained In MODMWARE.BAS**

Pauses for a specified number of seconds, allowing processes to occur in the background.

**Syntax**

```
Call MWPause(NumSecs!)
```

Where:

| | |
|---|---|
| *NumSecs!* | Specifies the number of seconds to pause execution. |

**Comments**

The *NumSecs!* argument is a floating point value, so you can pass fractions of a second. For example, if you wish to pause for a half of a second pass .5.

**NOTE** MWPause accepts values less than one second; however, its resolution under Windows is one timer tick, or approximately 1/18th of a second.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# MWPhoneBook

**Visual Basic Subroutine Contained In MWPHBOOK.BAS**

Displays a directory of phone numbers for dialing other modems.

**Syntax**

```
Call MWPhoneBook(Comm1)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |

**Comments**

Use MWPhoneBook to display a dialog box containing a directory of phone numbers. You can add new phone number entries to this directory. For each entry you can specify a name, phone number, baud rate, parity, data bits, stop bits, duplex (echo), handshaking, and a script file name. PDQComm saves all entries to the configuration file.

The user can dial any entry by clicking the Dial button in the dialog box. If there is a script file name specified for the dial routine, PDQComm executes the script. Otherwise PDQComm uses the standard dial routine, which redials if the line is busy.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, MWPHBOOK.BAS, MWPHBOOK.FRM, MWPROTCL.FRM, COMDLG32.OCX or COMDLG16.OCX, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# MWSetError

**Visual Basic Subroutine Contained In MODMWARE.BAS**

Sets the current ModemWare error.

**Syntax**

```
Call MWSetError(ErrorCode%)
```

Where:

| | |
|---|---|
| *ErrorCode%* | Specifies the error. |

**Comments**

ModemWare routines use the MWSetError to set the ModemWare error state. It is unlikely that you will need to call this routine.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# MWSetProtocol

**Visual Basic Function Contained In MWXFER.BAS**

Displays a dialog that allows the user to set the file transfer protocol.

**Syntax**

```
RtnValue = MWSetProtocol(Comm1)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |
| *RtnValue* | Returns True if the protocol is ASCII. False to all other supported protocols.. |

**Comments**

The MWSetProtocol function displays a dialog box with a list of supported file transfer protocols. When you select a protocol and press OK, PDQComm sets the protocol for the specified comm control. MWSetProtocol returns True for the ASCII protocol. The ASCII protocol transfers files verbatum, without any special processing.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, MWPROTCL.FRM, MWXFER.BAS, MWXFER.FRM, COMDLC32.OCX or COMDLG16.OCX, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# MWStatus

**Visual Basic Subroutine Contained In MODMWARE.BAS**

Returns the execution status for ModemWare routines. MWStatus creates a status message string, which you can display in a status bar.

**Syntax**

```
Call MWStatus(StatusCode%, ExtraData$)
```

Where:

| | |
|---|---|
| *StatusCode%* | Specifies the current status. |
| *ExtraData$* | Contains data relevant to the status. |

**Comments**

The last line in the MWStatus routine is usually a call to a StatusPrint routine, which displays the message devised in the MWStatus routine. See the CTRLDEMO.BAS file for an example of a StatusPrint routine.

Create your own StatusPrint routine to display a message on a status bar. You can create a status bar with an ordinary Visual Basic Picture control. To do so, set the Align property of the Picture control to 2 (align bottom), so that it will position itself along the bottom of your form, and set the AutoRedraw property to True, so that the contents of the picture control will not clear if the form repaints. Then, create a StatusPrint routine like the following:

```
Sub StatusPrint(Msg$) '-- Clear the picture control.
   Form1.Picture1.Cls '-- Print a space, and then the message.
   Form1.Picture1.Print " " & Msg$
End Sub
```

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# OpenPort

**Visual Basic Function Contained In MODMWARE.BAS**

Searches COM1: through COM9: for an available port and opens the first available port.

**Data Type**

Integer

**Syntax**

```
Port% = OpenPort(Comm1, PortNum%, BaudRate$, Settings$)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |
| *PortNum%* | Specifies the port number at which to start the search. |
| *BaudRate$* | Specifies the baud rate. |
| *Settings$* | Specifies the Parity, Data Bits, and Stop Bits (example: N,8,1). |
| *Port%* | Returns the port number that was opened, or 0 if no port was opened. |

**Comments**

OpenPort sets MWError () to one of the following values:

| **Value** | **Description** |
|---|---|
| MW_ER_CANCELED | Operation was Canceled. |
| MW_OPENPORT_OK | A port was opened. |
| MW_OPENPORT_SETTINGS | The specified settings are invalid. |
| MW_OPENPORT_ERROR | There are no available ports. |

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# PlaceCall

**Visual Basic Subroutine Contained In MWMODEM.BAS**

Places a call on the modem attached to the comm control and returns after a connection is made, the specified number of seconds has elapsed, or if the connection cannot be made (the line is busy, or some other error occurred).

## Syntax

```
Call PlaceCall(Comm1, Num$, Retries%, TimeOut%, Result%, ModemBaud&,
RealBaud&, Mode$)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |
| *Num$* | Specifies the phone number to dial including area code and prefixes. |
| *Retries%* | Specifies the number of attempts to make. |
| *TimeOut%* | Specifies the number of seconds to wait for a return code. |
| *Result%* | Returns a result code. |
| *ModemBaud&* | Returns the baud rate between the local computer and local modem. |
| *RealBaud&* | Returns the baud rate between our modem and the remote modem. |
| *Mode$* | Returns the rest of the CONNECT string after the baud rate. |

## Comments

The result code can be one of the following values:

| Value | Description |
|---|---|
| MW_RESULT_CONNECT | The modems are connected. |
| MW_RESULT_NO_CARRIER | The host modem did not answer. |
| MW_RESULT_ERROR | Modem Error. |
| MW_RESULT_NO_DIALTONE | The modem is not connected to the phone. |
| MW_RESULT_BUSY | The line is busy. |
| MW_RESULT_NO_ANSWER | The host modem did not answer. |
| MW_RESULT_TIMEOUT | The host modem did not answer. |

PlaceCall sets MWError () to one of the following values:

| Value | Description |
|---|---|
| MW_ER_CANCELED | Operation was canceled. |
| MW_ER_PORTCLOSED | The specified port does not exist. |
| MW_ER_CTRLERROR | The specified port can not be accessed. |
| MW_ER_TIMEOUT | Timeout. |
| MW_PLACECALL_OK | A valid result code was returned. |

When constructing a phone number, you can also insert special Hayes Compatible characters, called modifiers, into the phone number to do specific things. Here is a list of the standard modifiers:

| Modifier | Function |
|---|---|
| P | Dial with pulse modulation instead of touch tone. |
| W | Wait for dial tone. |
| , | Pause for n seconds (Number of seconds determined by the S8 register. The Default value is 2 seconds). |

| | |
|---|---|
| ; | Return to command mode after dialing |
| ! | Hook Flash. |

For example, if you need to dial a 9 and wait 2 seconds before dialing the number, use the following Num$ argument:

```
Num$ = "9,555-1212"
```

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, MWMODEM.BAS, MWMODEM.FRM, MWMODSTR.FRM, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# ScriptPlay

**Visual Basic Subroutine Contained In MODMWARE.BAS**

Play a script file.

**Syntax**

```
Call ScriptPlay(Comm1, FileName$, ErrorLineNum%, ErrorLine$)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |
| *FileName$* | Specifies the file to play. |
| *ErrorLineNum%* | Returns the line number at which an error occurred (if any). |
| *ErrorLine$* | Returns the script line at which an error occurred (if any). |

**Comments**

A script file is a text file that contains special ModemWare script commands that perform different communications tasks. You can use the ScriptRecord routine to record the script file to play.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, MWSCRIPT.BAS, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

**See Also**

ScriptRecord

# ScriptRecord

**Visual Basic Subroutine Contained In MWSCRIPT.BAS**

Record a script file.

**Syntax**

```
Call ScriptRecord(Comm1, FileName$, Length%)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |
| *FileName$* | Specifies the file to receive the text of the recorded script. |
| *Length%* | Specifies how many characters to record when a WAITFOR statement is recorded. |

**Comments**

To stop recording, call the ScriptRecord routine with the following syntax:

```
Call ScriptRecord (Comm1, "", 0)
```

To disable the ScriptRecord command, include the MWNOSCPT.BAS instead of the MWSCRIPT.BAS in your project.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, MWSCRIPT.BAS, MWNOSCPT.BAS, COMDLG32.OCX or COMDLG16.OCX, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

**See Also**

ScriptPlay

# SendModemCmd

**Visual Basic Subroutine Contained In MWMODEM.BAS**

Sends a command to the modem and returns the result code. See Hayes Modem Commands for more information about the modem commands that you can send with SendModemCmd.

**Syntax**

```
Call SendModemCmd(Comm1, Cmd$, NumTries%, TimeOut%, Result%)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |
| *Cmd$* | Specifies the command to send to the modem. |
| *NumTries%* | Specifies the number of tries to get an OK result code. |
| *TimeOut%* | Specifies The number of seconds to wait for a result code. |
| *Result* | Returns a result code. |

**Comments**

The result code can be one of the following values:

| **Value** | **Description** |
|---|---|
| MW_RESULT_OK | Modem accepted the command. |
| MW_RESULT_ERROR | Modem did not accept the command. |

SendModemCmd sets MWError () to one of the following values:

| **Value** | **Description** |
|---|---|
| MW_ER_CANCELED | Operation was Canceled. |
| MW_ER_PORTCLOSED | The specified port does not exist. |
| MW_ER_CTRLERROR | The specified port can not be accessed. |
| MW_ER_TIMEOUT | Timeout. |
| MW_SENDMODEMCMD_OK | A result code was returned. |
| MW_SENDMODEMCMD_NOCMD | No command was specified. |

The command (Cmd$) need not start with the attention string. Example: AT.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, MWMODEM.BAS, MWMODEM.FRM, MWMODSTR.FRM, MWPORT.BAS, MWPORT.FRM, MWSCRIPT.BAS, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# SetPortSettings

**Visual Basic Subroutine Contained In MWPORT.BAS**

Displays a dialog that allows users to configure a comm port and save the settings to the configuration file.

**Syntax**

```
Call SetPortSettings(Comm1)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |

**Comments**

You do not have to call SetPortSettings directly unless the user wants to change his or her port setup. LoadPortSettings automatically calls SetPortSettings if it cannot find an entry in the configuration file or if the file does not exist.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, MWPORT.BAS, MWPORT.FRM, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# SetTerminalSettings

**Visual Basic Subroutine Contained In MWTERM.BAS**

Displays a dialog that allows users to configure the terminal window and save the settings to the configuration file.

**Syntax**

```
Call SetTerminalSettings(Comm1)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |

**Comments**

You do not have to call SetTerminalSettings directly unless the user wants to change his or her terminal settings. LoadTerminalSettings automatically calls SetTerminalSettings if it cannot find an entry in the configuration file or if the file does not exist.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, MWTERM.BAS, MWTERM.FRM, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# SetModemSettings

**Visual Basic Subroutine Contained In MWMODEM.BAS**

Displays a dialog that allows users to select modems from the ModemWare modem database and save the information for a selected modem to the configuration file.

**Syntax**

```
Call SetModemSettings(Comm1)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |

**Comments**

You do not have to call SetModemSettings directly unless the user wants to change his or her modem setup. LoadModemSettings automatically calls SetModemSettings if it cannot find an entry in the configuration file or if the file does not exist.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, MWMODEM.BAS, MWMODEM.FRM, MWMODSTR.FRM, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# Upload

**Visual Basic Subroutine Contained In MWXFER.BAS**

Uploads a file, letting the user pick the file transfer protocol and filename.

**Syntax**

```
Call Upload(Comm1)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |

**Comments**

Upload displays the protocol dialog box, which contains a list of all file transfer protocols supported by PDQComm. If the user picks a protocol that requires the specification of a file name, PDQComm displays the file dialog box. If the protocol does not require a file name (Ymodem-G, Ymodem-Batch, Zmodem, Compuserve B+) PDQComm does not display the file dialog box. The upload occurs in the background, during which PDQ_EV_XFER events fire via the OnComm event. When one of these events occurs, the XferStatus property logs the status of the transfer.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS MWXFER.BAS, MWXFER.FRM, MWPROTCL.FRM, COMDLC32.OCX or COMDLG16.OCX, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# WaitFor

**Visual Basic Function Contained In MODMWARE.BAS**

Waits for a specified number of seconds to receive a specific string over the serial port.

**Data Type**

String

**Syntax**

```
Received$ = WaitFor$(Comm1, Wait$, TimeOut%, CDFlag, ExcludeFlag)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |
| *Wait$* | Specifies the string to wait for. |
| *TimeOut%* | Specifies the number of seconds to wait. |
| *CDFlag* | Set to True if you wish WaitFor$ to return if the carrier is lost. |
| *ExcludeFlag* | Set to True if you do not wish WaitFor$ to return the specified string as part of the received data. |

**Comments**

WaitFor$ sets MWError () to one of the following values:

| Value | Description |
|---|---|
| MW_ER_CANCELED | Operation was cancelled. |
| MW_ER_LOST_CARRIER | Carrier was lost. The call was disconnected. |
| MW_ER_PORTCLOSED | The specified port does not exist. |
| MW_ER_CTRLERROR | The specified port can not be accessed. |
| MW_ER_TIMEOUT | Timeout. |
| MW_WAITFOR_OK | No Errors. |

Setting ExcludeFlag to True tells WaitFor$ to return everything that was received up to but not including Wait$.
Setting ExcludeFlag to False tells WaitFor$ to return everything that was received, including Wait$.

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# WaitForA

**Visual Basic Function Contained In MODMWARE.BAS**

Waits for a specified number of seconds to receive one of several strings over the serial port.

**Data Type**

String

**Syntax**

```
Received$ = WaitForA$(Comm1, Wait$(), TimeOut%, CDFlag, ExcludeFlag, Found)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control. |
| *Wait$()* | Specifies an array of strings to wait for. |
| *TimeOut%* | Specifies the number of seconds to wait. |
| *CDFlag* | Set to True if you wish WaitForA$ to return on carrier loss. |
| *ExcludeFlag* | Set to True if you do not wish WaitForA$ to return the specified string as part of the received data. |
| Found | Returns the element number of the found string, or -1 if a timeout occurred. |
| Received$ | Returns everything that was received since WaitForA$ was called. |

**Comments**

WaitForA$ sets MWError () to one of the following values:

| Value | Description |
|---|---|
| MW_ER_CANCELED | Operation was cancelled. |
| MW_ER_LOST_CARRIER | Carrier was lost. The call was disconnected. |
| MW_ER_PORTCLOSED | The specified port does not exist. |
| MW_ER_CTRLERROR | The specified port can not be accessed. |
| MW_ER_TIMEOUT | Timeout. |
| MW_WAITFOR_OK | No Errors. |

Setting ExcludeFlag to True tells WaitForA$ to return everything that was received up to but not including Wait$ (x). Setting ExcludeFlag to False tells WaitForA$ to return everything that was received, including Wait$ (x).

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

# WaitForCall

**Visual Basic Subroutine Contained In MWMODEM.BAS**

Waits for a call on an open port to which a modem is attached.

**Syntax**

```
Call WaitForCall(Comm1, NumRings%, ModemBaud&, RealBaud&, Mode$)
```

Where:

| | |
|---|---|
| *Comm1* | Specifies the communications control |
| *NumRings%* | Specifies the number of rings to answer on. |
| *ModemBaud&* | Returns the baud rate between the local computer and the local modem |
| *RealBaud&* | Returns the baud rate between the local modem and the remote modem |
| *Mode$* | Returns the rest of the CONNECT string after the baud rate |

**Comments**

WaitForCall sets MWError () to one of the following values:

| **Value** | **Description** |
|---|---|
| MW_ER_CANCELED | Operation was Canceled |
| MW_ER_PORTCLOSED | The specified port does not exist. |
| MW_ER_CTRLERROR | The specified port can not be accessed |
| MW_ER_TIMEOUT | Timeout |
| MW_WAITFORCALL_OK | We are connected to the caller. |

**Required Files**

COMMCTRL.BAS, MODMWARE.BAS, MWMODEM.BAS, MWMODEM.FRM, MWMODSTR.FRM, a PDQComm control file (PDQCOM32.OCX, PDQCOM16.OCX)

#  :LABEL Command

Specifies a label. This works the same as with batch files. Any word that starts with a colon is treated as a label.

**Syntax**

`:`*`label`*

**Example**

```
:JumpHere
```

# CAPTURE Command

Opens a specified capture file and logs all received data to a specified file.

**Syntax**

`CAPTURE` *`filename$`*

**Example**

```
CAPTURE "C:\CSERVE.TXT"
```

**See Also**

CLOSECAPTURE

# CLOSECAPTURE Command

Closes the capture file opened previously with CAPTURE.

**Syntax**

```
CLOSECAPTURE
```

**See Also**

CAPTURE

#  DIAL Command

Dials a phone number and connects to a host modem.

**Example**

```
DIAL "555-1212", 10
WAITFOR "Press Enter to Continue" SEND "^M"
```

This example dials 555-1212. The second argument (10) specifies the number of retries if the line happens to be busy. After a connection is made, it waits to receive Press Enter to Continue and then sends a carriage return.

**See Also**

:LABEL, ON_TIMEOUT_GOTO, WAITFOR, SEND.

# DOWNLOAD Command

Downloads (receives) a file.

### Syntax

```
DOWNLOAD [filename$]
```

### Examples

```
'-- Receives the file C:\MYPROG\ZIPFILE.ZIP via Xmodem-CRC.
PROTOCOL "XMODEM-CRC"
DOWNLOAD "C:\MYPROG\ZIPFILE.ZIP"
```

```
'-- Receives a file via ZMODEM.
PROTOCOL "ZMODEM"
DOWNLOAD
```

**NOTE** With the ZMODEM protocol, the file name is not specified. This is because the remote side of a zmodem transfer session does not require a filename. This is also true for YMODEM- BATCH, and YMODEM-G.

### See Also

PROTOCOL

# END Command

Ends the script. END does not close the communications port.

**Syntax**

```
END
```

# HANGUP Command

Hangs up the phone. Hangup does not close the communications port, it only disconnects the modem.

**Syntax**

```
HANGUP
```

# INPUT Command

Prompts the user for a string and stores the data as a string in a variable.

**Syntax**

```
INPUT msgStr$, variableName
```

**Example**

```
INPUT "Enter Your Password", PassWord
SEND PassWord
SEND "^M"
```

This example displays an input dialog box with the message; Enter Your Password and stores the entered text in a variable named PassWord. Then, that text is sent out the communications port followed by a carriage return.

**See Also**

SEND

# ON_TIMEOUT_GOTO Command

Tells the script to jump to a label when a timeout occurs. Several things can cause a timeout to occur; a busy signal on the other end of the line, accessing an inactive modem, etc.

**Syntax**

```
ON_TIMEOUT_GOTO :label
```

**Example**

```
ON_TIMEOUT_GOTO :JumpHere
```

**See Also**

:LABEL

# PAUSE Command

Pauses for a specified number of seconds.

**Syntax**

```
PAUSE n%
```

**Example**

```
PAUSE 1  '-- Wait one second.
```

**NOTE** Sometimes a pause is required in between a WAITFOR and a SEND command, or in other places to smooth out the communications process. This may require experimentation.

**See Also**

WAITFOR, SEND

# PORT Command

Specifies which port to use and opens the port. By default, scripts use the current port with its current settings.

**Syntax**

```
PORT n%
```

**Example**

```
PORT 2    '-- Use COM2.
```

# PROTOCOL Command

Sets the file transfer protocol.

**Syntax**

```
PROTOCOL protocol$
```

*protocol$* is one of the following string identifiers:

| | |
|---|---|
| XMODEM-CHECKSUM | YMODEM-G |
| XMODEM-CRC | ZMODEM |
| XMODEM-1K | KERMIT |
| YMODEM-BATCH | COMPUSERVE |

**NOTE** You must issue this command to set the protocol before either a DOWNLOAD or UPLOAD command. If you do not, the script uses the current protocol setting for the PDQComm control.

**Example**

```
PROTOCOL "ZMODEM"
```

**See Also**

DOWNLOAD, UPLOAD

# SEND Command

Sends a string out of the communications port. The Send command treats an unquoted string as a variable, sending the text in the variable.

You can embed control codes in the text. These are defined in ASCII as ^A through ^Z corresponding to ASCII values 1 through 31 respectively. Thus, ^G is a beep, ^M is a carriage return, ^J is a linefeed, etc.

**Examples**

```
SEND "Hello Over There^M"  '-- Sends the string followed by a
                           '   carriage return.

'-- This example asks the user for his/her name, and sends
' it followed by a CR/LF.
INPUT "Enter Your Name", UserName
SEND UserName
SEND "^M^J"
```

**See Also**

INPUT

# SETTINGS Command

Specifies the baud rate, parity, data bits, and stop bits. By default, scripts use the current port with its current settings if the communications port is opened.

**Syntax**

```
SETTINGS string$
```

The setting string is composed of four settings and has the following format:

```
"BBBB,P,D,S"
```

Where *BBBB* is the baud rate, *P* is the parity, *D* is the number of data bits, and *S* is the number of stop bits.

**Example**

```
SETTINGS "9600,N,8,1"
```

# STOP Command

Stops program execution. Use this command to debug your scripts. When a script encounters the STOP command, Visual Basic stops program execution. If you are running your program as an executable, the program exits.

**Syntax**

```
STOP
```

# TIMEOUT Command

Specifies the number of seconds that WAITFOR waits before timing out.

**Syntax**

```
TIMEOUT n%
```

**Example**

```
TIMEOUT 30
```

**See Also**

WAITFOR

# UPLOAD Command

Uploads (transmits) a file.

**Syntax**

```
UPLOAD [filename$]
```

**Examples**

```
'-- Sends the file C:\MYPROG\ZIPFILE.ZIP via Zmodem.
PROTOCOL "ZMODEM"
UPLOAD "C:\MYPROG\ZIPFILE.ZIP"

'-- Sends a file via Compuserve B+.
PROTOCOL "COMPUSERVE"
UPLOAD
```

**NOTE** With the Compuserve protocol, the file is not specified. This is because the remote side of a Compuserve transfer session does not require a filename.

**See Also**

PROTOCOL

#  WAITFOR Command

Waits to receive a string over the communications port.

**Syntax**

```
WAITFOR string$
```

**Example**

```
TIMEOUT 30
DIAL "555-1212"
WAITFOR "What Is Your Name?"
SEND "ModemWare^M"
```

This example dials 555-1212 and waits up to 30 seconds (specified by TIMEOUT) after connecting to receive What Is Your Name? from the host system, and sends ModemWare followed by a carriage return.

**See Also**

:LABEL, ON_TIMEOUT_GOTO, TIMEOUT

# Hayes "AT" Modem Commands

Related Topics

All Hayes-compatible modems recognize a standard set of commands, to control various aspects of the modems operation. You can use the ModemWare SendModemCmd routine to send a command to a modem. Most of these commands begin with the letters AT, which stands for Attention. For example, the following command tells the modem to access the phone line and dial a number.

```
ATDT
```

This topic contains a listing of the most commonly used modem commands. See your modem manual for a complete listing of modem commands and descriptions of advanced commands supported by your modem.

**NOTE** All commands must be followed by a Chr$(13) carriage return, except +++and A/. Also, many modems require these commands to be sent in upper case.

For more information about modems and modem commands, see the MODEMBAS.WRI file located in the PDQComm product directory.

| Command | Function | Description |
|---|---|---|
| +++ | Command mode | Returns the modem to command mode from online mode. Online mode is when you are connected to a remote system. In order for your modem to recognize the escape sequence, you must not send any characters to the modem for at least 1 second before and after the escape sequence. This is generally used to get the modems attention so you can send an ATH command to force it to hang up the line. The modem responds with OK when it goes into command mode. At that point you may issue any of the AT commands. |
| ATO | Online mode | Returns the modem to on-ine mode after the +++ escape sequence has put it into command mode. Note that the O is an upper case letter, and not a zero. |
| A/ | Execute last command sent | A/ does not need a terminating carriage return. |
| ATA | Answer phone | Tells the modem to pick up the phone and attempt to connect with a remote modem. |
| ATD*x* | Dial a telephone number | The x is replaced with P for pulse dialing or T for touch tone dialing. For example, to call the Crescent Software Support BBS, you would issue the following string to the modem:<br><br>"ATDT1-617-280-4221" + Chr$(13)<br><br>If you have rotary (pulse) dialing, you would replace the T with a P. If your phone is on a PBX system and requires you to dial 9 and wait for a dial tone, you can add a comma between the 9 and the rest of the number as shown below:<br><br>"ATDT9,1-617-280-4221" + Chr$(13) |

| | | |
|---|---|---|
| | | The comma specifies a 2 second pause. For longer pauses simply use multiple commas. You may also use the S-8 register to change the delay time for each comma received. (See S-registers) |
| ATE*n* | Echo all commands to the local display | The n is replaced by either 0 or 1. ATE0 disables character echo in command mode, and you will not see subsequent AT commands on your PC as you type. ATE1 enables command echo to the screen, which is the default for most modems. Typing ATE alone is the same as ATE0. |
| ATH*n* | Hook switch control | The n value is either 0 or 1. ATH0 causes the modem to hang up (go on-hook). ATH1 causes the modem to pick up the line (go off-hook). ATH1 is not the same as ATA because the modem will not try to connect when ATH1 is used. You could use ATH1 if you want your BBS to return a busy signal to callers. Using ATH alone is equivalent to ATH0. |
| ATM*n* | Speaker control | The n values range from 0 through 2. ATM0 keeps the speaker turned off at all times, ATM1 keeps the speaker on until a carrier is detected, and ATM2 keeps the speaker on at all times. The default for most modems is ATM1. |
| ATQ*n* | Result code control | The n value is either 0 or 1. ATQ0 tells the modem to echo the result of each AT command to the local display, and ATQ1 disables echo. In most cases you will not want to disable the result codes because they can be used by your programs to determine the success or failure of an operation. The default for most modems is ATQ0. |
| ATS*r*? | Read S-Register | S-registers control many modem functions and are described below. The r is replaced with the register number you want to read. For example, to read S-register 0 you would send the command string ATS0? The modem then returns the current setting to the console for display. |
| ATS*r*=*n* | Set S-Register | Set a specified S-register r to the value you assign as n. For example, sending ATS8=1 tells the modem to pause only one second for each comma received instead of two. |
| ATV*n* | Select result code format | The n value is either 0 or 1. ATV0 selects the short form of result codes, which display either one or two digits. ATV1 specifies the long form of result code. Long codes return a text string that describes the result in words. See Modem Command Result Codes for a listing of common result codes and their descriptions. The default for most modems is ATV1. |
| ATX*n* | Extended result code selection | This command means different things with different modems. The n value tells the modem how many different result codes you want. For example, ATX0 restricts the range to 300 BPS-compatible result codes, which is only CONNECT. Most modems use ATX4 to enable |

| | | |
|---|---|---|
| | | all result codes, but you should check your modem manual to be certain. |
| ATZ | Modem reset command. | Restores the modem to the power-on default configuration set in ROM or non-volatile RAM. Depending on your modem brand and model, you may be able to store selected modem default settings that are retained even when the power is turned off. |

# Modem Command Result Codes

This topic list some of the common result codes returned by Hayes modem commands.

| Short Form | Long Form | Description |
|---|---|---|
| 0 | OK | Command received okay |
| 1 | CONNECT | Connect at 300 BPS |
| 2 | RING | Ring detected |
| 3 | NO CARRIER | No carrier detected |
| 4 | ERROR | Error in command |
| 5 | CONNECT 1200 | Connect at 1200 BPS |
| 6 | NO DIAL TONE | No dial tone found |
| 7 | BUSY | Busy signal |
| 8 | NO ANSWER | No answer |
| 10 | CONNECT 2400 | Connect at 2400 BPS |

# S-Registers

The S-Registers define various modem configuration settings and are specified using the ATS command. For example, register S0 lets you specify how many times the phone must ring before the modem is to answer it. You may read and write these registers, using the ATS command described in the topic on Hayes modem commands.

| Register | Function | Description |
|---|---|---|
| S0 | Ring to answer | Specifies the number of times the phone must ring before the modem will pick up. If set to 0, auto answer is disabled. |
| S1 | Ring count | Counts the number of rings. Resets to 0 if a ring is not sensed within any 8-second interval. S1 operates only when S0 is set to a value greater than 0. |
| S2 | Escape code character | Sets the ASCII value of the character used to send an escape sequence. The default is 43 (+). |
| S3 | Carriage return character | Sets the ASCII value of the character recognized as a carriage return. The default is 13. |
| S4 | Line feed character | Sets the ASCII value of the character recognized as a line feed. The default is 10. |
| S5 | Backspace character | Sets the ASCII value of the character recognized as a backspace. The default is 8. This register is limited to values between 0 and 32. |
| S6 | Wait for dial tone | Sets the number of seconds the modem should wait for a dial tone before returning a NO DIAL TONE message. |
| S7 | Wait for carrier after dial | Sets the number of seconds the modem should wait for a carrier after dialing before issuing a NO CARRIER message. |
| S8 | Pause time for comma | Sets the number of seconds to pause for each comma encountered within an ATD (dial) command. The default is 2 seconds. |
| S9 | Carrier detect response time | Sets the number of seconds a received carrier must be present for before the modem will recognize it as a valid carrier signal. |
| S10 | Lost carrier to hang up delay | Sets the number of seconds the modem should wait before hanging up the line when the carrier is lost. |
| S25 | Delay to DTR | This is the number of seconds the DTR line must be false before the modem is to hang up the line. |

Modem Command Result Codes
S-Registers
SendModemCmd