



PowerTCP Version 3.0

[General Information on PowerTCP](#)

[PowerTCP Tutorial](#)

[Frequently Asked Questions](#)

[Additional Resources](#)

<b>Protocol</b>	<b>Overview</b>	<b>Reference</b>
TCP - Transmission Control Protocol	<input type="checkbox"/>	
<input type="checkbox"/>		
Telnet - Telecommunications Protocol	<input type="checkbox"/>	
<input type="checkbox"/>		
VT Emulation	<input type="checkbox"/>	
<input type="checkbox"/>		
FTP - File Transfer Protocol	<input type="checkbox"/>	
<input type="checkbox"/>		
SMTP - Simple Mail Transfer Protocol	<input type="checkbox"/>	
<input type="checkbox"/>		
POP3 - Post Office Protocol	<input type="checkbox"/>	
<input type="checkbox"/>		
UDP - User Datagram Protocol	<input type="checkbox"/>	
<input type="checkbox"/>		
SNMP - Simple Network Management	<input type="checkbox"/>	
<input type="checkbox"/>		
TFTP - Trivial File Transfer Protocol	<input type="checkbox"/>	
<input type="checkbox"/>		

## **General Information on PowerTCP**

PowerTCP is a set of TCP/IP communications libraries that you can quickly utilize to build powerful TCP/IP applications. Each library hides the details of the communications protocol from the calling application, providing an explicit and easy-to-use interface.

PowerTCP provides network communications using the TCP/IP protocol suite, automatically taking care of many details of socket library programming, including:

- Setting up connections (creating sockets, resolving names, making active and passive connections, etc.)
- Closing connections (destroying sockets and buffers, etc.)
- Data buffering and application flow control
- Safe error recovery
- Optimized design with event notification
- Tested operation on numerous Windows Sockets implementations
- Upper-layer features depending on the application-layer protocol implemented

### PowerTCP Interfaces

#### IMPORTANT: PowerTCP and Event Processing

#### The Windows Socket Interface and PowerTCP

#### Introduction to Asynchronous Event Notification

#### Flow Control

## **PowerTCP Interfaces**

### General Information

PowerTCP provides five interfaces in the form of:

- C++ class libraries from which user classes are derived. This requires Microsoft C++ compiler compatibility because this interface is supplied as static linked libraries. 16-bit versions are provided in all models, and a 32-bit version is provided for Windows NT and Windows 95.
- Dynamic Link Libraries, for use with C/C++, or any other language which supports DLLs. 16 and 32-bit versions are provided.
- Visual Basic custom controls which provide an easy-to-use interface to TCP through properties and events.
- Delphi VCL Components, for use in Borland's Delphi environment.
- OLE controls for many high-level development environments.

All PowerTCP libraries have an identical internal architecture. C++ is the native mode provided in library form, with C, Visual Basic, Delphi, and OLE interfaces added and compiled as separate .DLL, .VBX, .DCU and .OCX files.

## PowerTCP and Event Processing

General Information

### **IMPORTANT**

Visual Basic was designed to protect your program from re-entrancy by only allowing one event to be fired at any one time. Consequently, if you put a debugging break point inside a PowerTCP event, no other events will be fired. Similarly, if you use a MsgBox to display a value within a PowerTCP event, all other communication events will be disabled while the MsgBox captures the thread of execution. Also, if a DoEvents statement is placed within any Visual Basic event, all PowerTCP communication events will be disabled by Visual Basic.

Therefore, to effectively debug your communications program, you should capture values by putting them into an edit box or some other control that does not trap your thread within an event handler. Breakpoints and MsgBox's can interrupt the data stream you are trying to work with.

### **CRITICAL**

**NEVER** use the DoEvents statement within a Visual Basic event, as all PowerTCP event notification will be disabled.

## The Windows Socket Interface and PowerTCP

### General Information

PowerTCP libraries lie between your application and the Windows Sockets interface (WINSOCK.DLL and/or WSOCK32.DLL). This section describes exactly how these components interact.

Figure 1-1 below illustrates the Standard Open Systems Interconnect (OSI) 7-layer communications model. In the middle is an illustration of how TCP/IP maps to this model, and to the right we show how PowerTCP libraries fit into the picture.



**Figure 1-1. Standard OSI 7-layer communications model and PowerTCP**

The computer supplies the hardware (Physical Layer) necessary to make data communications work. Typically, this is an RS-232 port or a network interface card (ethernet, token-ring, etc.), and often includes the Data Link Layer (media access control or MAC and logical link control or LLC found on most network interface cards).

Interfacing to the Data Link Layer is the Network Layer where the IP (Internet Protocol) communication processing takes place. This interface can be implemented using de-facto standards like the Network Device Interface Standard (NDIS) or Open Data-link Interface (ODI).

TCP and UDP protocol processing is usually considered as operating within the Transport Layer, and is normally implemented by the same vendor that provides the IP processing. Consequently, there is no standard interface between the Network and Transport Layers when it comes to TCP/IP.

A standard interface is necessary, however, between the Application Layer where the Upper-Layer Protocol (Telnet, FTP, SMTP, TFTP, etc.) is implemented and the Transport Layer that is provided by the TCP/IP vendor. This enables third-party developers to write TCP/IP applications. Currently, under UNIX and Windows, vendors provide static libraries (the "socket library") that allow application developers to utilize TCP/IP services. Recently, however, the advent of dynamically-linked libraries under Windows enabled the industry to standardize on a Windows socket library standard, which is now in effect, and is called the "Windows Sockets" interface (version 1.1 is current as of this writing). This standard allows our libraries to run on many TCP/IP transports, greatly increasing the market for TCP/IP applications written for Windows.

All PowerTCP products interface to the Windows Sockets interface, and rely on it for operation. PowerTCP adds value to the Windows Sockets interface by encapsulating all library calls in an optimized fashion. PowerTCP provides:

1. A completely event-driven, turn-key design. By using PowerTCP libraries, you take advantage of an optimized design with few entry points. Little design work is required within the application.
2. Accelerated development. By following the examples provided, application development is normally accelerated by months. Only an understanding of the upper-layer protocol functionality is necessary, without requiring detailed understanding of the Windows Sockets interface/behavior.
3. Risk management. Tested components reduce risk and unexpected project delays.

PowerTCP provides these features and upper-layer protocols through all supported interfaces.



## Introduction to Asynchronous Event Notification

### General Information

There are different kinds of methods used to notify programs when communication events occur. The simplest type is *synchronous blocking*. Disk access under DOS, for example, is synchronous blocking because the **Read()** function does not return until it has data from the disk file. In other words, program execution is blocked by the I/O action until it completes. This may be acceptable when dealing with high-speed local access, but synchronous blocking is inappropriate when the I/O action may involve a TCP/IP host on the other side of the world.

A second type is *synchronous non-blocking*. Using this mechanism, one could launch a request to read the disk, and check back later to see if it completed successfully. The program could do other things while the I/O occurs, enhancing response to user input, etc. The drawback to this approach, however, is that the application must set a timer and poll for completion. Several checks may take place before the I/O is complete, and when it finishes there is wasted time until the program checks again.

A third type is *asynchronous*. This mechanism is preferred for the following reasons:

- A program spends only a small amount of time in the function
- No inefficient polling mechanism is needed
- Immediate event notification produces faster response times

One of the most advanced and distinguishing features of the PowerTCP line is that it uses asynchronous I/O for all communications. The Connect process, for example, sends the host name specified to the *DNS* (Domain Name Server) and immediately returns control to the application. After the DNS replies with the resolved address, a connection request is made. When that completes, the application is notified of the connection event using the Visual Basic event mechanism. The custom control has a Connect event that PowerTCP calls when a successful connection is created. Similarly, event notification occurs for all other significant events that occur during the communications session.

This type of event-driven communications code results in very concise code with low structural overhead.

## Flow Control

### General Information

To maximize data throughput, you should understand the flow control mechanisms provided by PowerTCP. This section describes the flow control issue for the sending and receiving of data.

### **Sending Data**

Most communications interfaces are polling-oriented, where the user attempts to send  $x$  bytes and gets feedback from the system that  $y$  bytes were accepted by the system buffers. The Windows Sockets interface and the Windows asynchronous interface functions behave in this manner. This implies that you must set up some type of incremental feeding mechanism in your application.

PowerTCP does this for you, so there must be an event that notifies your program when the system has accepted your data. This is **Send** event, which is called when your buffer has been completely accepted by the system buffer. For applications that must maximize transmission rates, you should submit large buffers using the **Send** property upon notification of each **Send** event for the previous buffer. This will maximize output without requiring excessive buffer memory.

### **Receiving Data**

When receiving large buffers of data, we recommend that you process the data immediately (for most applications). If this involves significant processing, then the TCP receive buffers will fill and the local host will effectively slow the transmission rate from the sender of the data. This is called backpressure and is the best way to compensate for high loading at your end of the connection. PowerTCP also provides an alternative option for users who want to receive data only when polled - the **Recv** property signals PowerTCP it is time to read from the network buffers, and limits the size of received buffers to an arbitrary count (setting to 0 turns off automatic receiving of data). This technique can be used to increase backpressure if desired.

### **Closing TCP and UDP Connections**

The **Recv** event occurs with `RecvData = ""` when the connection is terminated.

### **NOTE**

PowerTCP uses messages from the Winsock interface to generate receive and send activity. Each Winsock receive message causes data to be read into a buffer and passed to the user's application. Each Winsock send event causes PowerTCP to fill the send buffers with as much data as can be accepted at that time.



## PowerTCP Tutorial

### The Application

The sample application created in this chapter is similar to the one included with PowerTCP, accessible in the PowerTCP program group. You can run the sample by clicking on its icon, or open the Visual Basic project by clicking on its icon.



The sample created in this chapter differs only in minor ways, such as the absence of an About box.

Step 1: Add the necessary files

## Step 1: Add the necessary files

The first step is to add the necessary files to the project.



First, add the **POWERTCP.BAS** file from the \POWERTCP\INCLUDE directory to your project. This file defines all necessary constants for PowerTCP, which makes your code more readable.

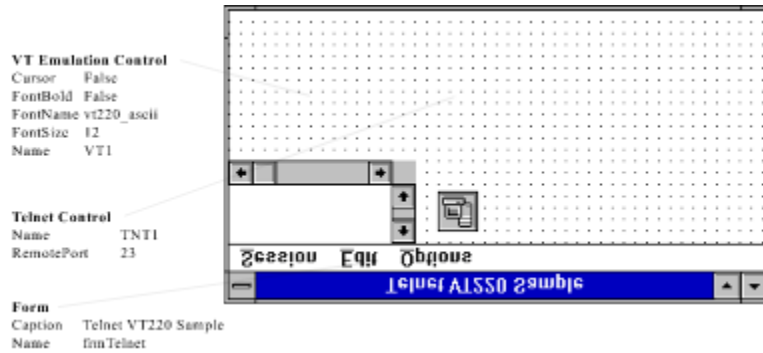


Next, add both the **P16TNTB5.VBX** and **P16VT2B5.VBX** files to your project from your \WINDOWS\SYSTEM directory. These are the Telnet and VT emulation custom controls, respectively.

Step 2: Create the user interface

## Step 2: Create the user interface

The next step is to place the appropriate controls and menus on the appropriate forms. Figure 3-1 below shows the main application window, with the property settings for each control:

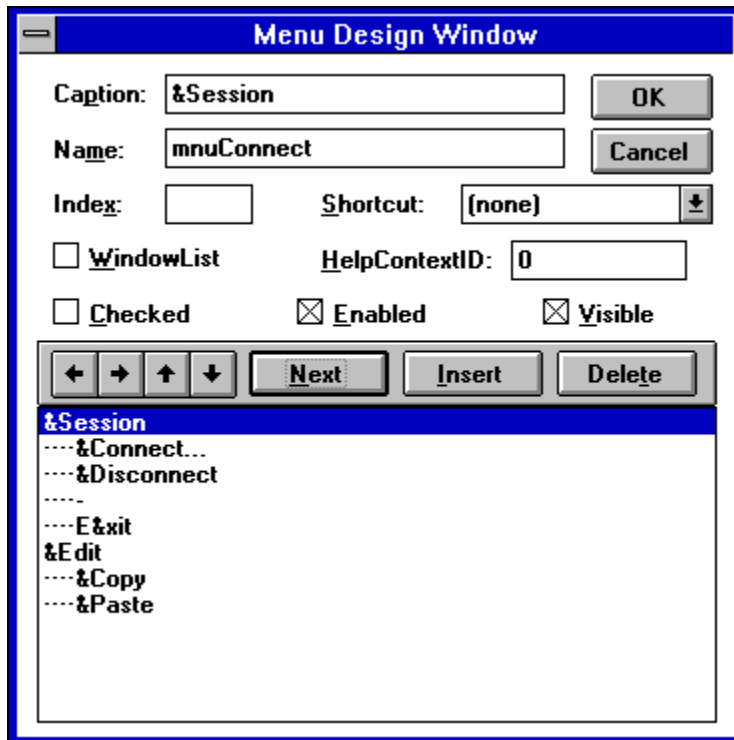


**Figure 3-1. The main application window.**

Next: Add the Menus

## Add the Menus

The menus for the window are shown below in Figure 3-2:



The screenshot shows a 'Menu Design Window' with the following fields and controls:

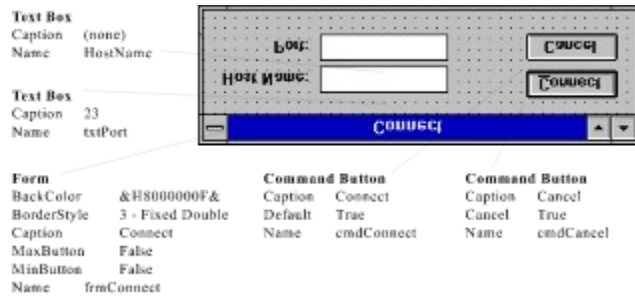
- Caption:**
- Name:**
- Index:**  **Shortcut:**
- WindowList** **HelpContextID:**
- Checked**  **Enabled**  **Visible**
- Navigation buttons:
- Menu List:**
  - &Session**
  - &Connect...
  - &Disconnect
  - 
  - E&xit
  - &Edit**
  - &Copy
  - &Paste

**Figure 3-2. The menu design window.**

Next: Add a Second Form

## Add a Second Form

Another form must be added to the project, which will be brought up when the user chooses the **Connect...** from the **Session** menu. The form appears in Figure 3-3:



**Figure 3-3. Adding a second form.**

Step 3: Write the code

### **Step 3: Write the code**

This section lists and explains the code needed for a Telnet application.

[Next: Make the Connection](#)

### Make the Connection

The following code should be placed in the event handler of the **Session/Connect...** menu:

```
Sub mnuCRemoteSystem_Click ()
    frmConnect.Show 1
End Sub
```

This displays the connection form.

The following code should be placed in the events handler of the **Connect** button on the Connection form:

```
Sub cmdConnect_Click ()
    ' Set the port (this is normally 23)
    frmTelnet.TNT1.RemotePort = Val(txtPort)

    ' Tell the control where to connect to
    frmTelnet.TNT1.RemoteHost = HostName

    ' Tell the control to connect
    frmTelnet.TNT1.Action = CONNECTCOMM

    ' Hide this form
    Unload Me
End Sub
```

When the connection is established, enable the VT control and show its cursor:

```
Sub TNT1_Connect ()
    VT1.Enabled = True
    VT1.Cursor = True
End Sub
```

This provides UI feedback to the user. If the user had clicked **Cancel**, the form would be unloaded:

```
Sub cmdCancel_Click ()
    Unload Me
End Sub
```

Next: Disconnect

## Disconnect

The code for closing communications is shown here. It should be placed in the **Session/Disconnect** event:

```
Sub mnuCDisconnect_Click ()  
    ' Close the connection  
    TNT1.Action = CLOSECOMM  
End Sub
```

Next: Transfer the data over the connection



### Transfer the data

When data arrives over the connection, place it in the VT control. The arrival of a zero-length string indicates the connection has been closed.

```
Sub TNT1_Recv (RecvData As String)
  If RecvData = "" Then
    ' Let the user know the connection is down
    VT1.Cursor = False
  Else
    ' Place the received characters into the VT control
    VT1.Display = RecvData
  End If
End Sub
```

Similarly, when a key is pressed in the VT control, respond by sending it to the Telnet control:

```
Sub VT1_KeyPress (KeyString As String)
  ' Send the character to the host
  TNT1.Send = KeyString
End Sub
```

Next: Option Negotiation

## Option Negotiation

Option negotiation is the process where the two ends of the Telnet connection try to determine what features they will and won't support. This program will support these features:

- Host echoing
- Suppressing go-aheads (leads to faster transfers)

The Telnet custom control supports two types of option negotiation - automatic and customizable. In this example, automatic option negotiation is selected by setting the AutoOption property on the Telnet control to True at design time (this is the default value).

### **MORE INFO**

The following code is **skeleton code** for handling option negotiation. Each occurrence of *Option\_you\_wish\_to\_support* should be replaced by a Telnet option negotiation constant from POWERTCP.BAS. For example, a common option is SUPPRESS\_GO\_AHEADS.

```

Sub TNT1_Cmd (Cmd As Integer, TelnetOption As Integer, SubOption As String)
Select Case Cmd

Case DO_CMD
' The host wants us to support something
If TelnetOption = Option_you_wish_to_support Then
' The host wants to negotiate the option
TNT1.WillOption = Option_you_wish_to_support
ElseIf TelnetOption = Other_Option_you_wish_to_support Then
' The host wants to negotiate the option
TNT1.WillOption = Other_Option_you_wish_to_support
Else
' We won't support any other options
TNT1.WontOption = TelnetOption
End If

Case SB_CMD
' The host requires more information about
' a feature

' Negotiate the Option_you_wish_to_support suboption
If TelnetOption = Option_you_wish_to_support Then
TNT1.SubOption = Additional_option_infomation
TNT1.DoSubOption = Option_you_wish_to_support
End If

Case DONT_CMD
' The host wants us to not do something

If TelnetOption = Option_you_wish_to_support or TelnetOption
= Other_Option_you_wish_to_support Then
' We only support those options
TNT1.WontOption = TelnetOption
End If

Case WILL_CMD
' The host will do something and is asking
' for permission

If TelnetOption = Option_you_wish_to_support Then
' Tell the host he can do it
TNT1.DoOption = Option_you_wish_to_support
ElseIf TelnetOption = Other_Option_you_wish_to_support Then
' Tell the host he can do it
TNT1.DoOption = Other_Option_you_wish_to_support
Else
' We don't support any other options
TNT1.DontOption = TelnetOption
End If

Case WONT_CMD
' If the host won't do something, then we will
' confirm it, because we have to.

TNT1.DontOption = TelnetOption
End Select

```

End Sub

**Next: Edit Commands**

## Edit Commands

This terminal emulator program will support two edit commands - Copy and Paste. Copy will copy selected text into the clipboard, and Paste will send the text in the clipboard to the connected Telnet host. The code below shows how easy this is:

```
Sub mnuECopy_Click ()
    ' Copy the selected text to Clipboard by clearing
    ' it and using the Mid$ function to grab the selected
    ' chunk of text

    Clipboard.Clear
    Clipboard.SetText Mid$(VT1.Text, VT1.SelStart, VT1.SelLength)
End Sub

Sub mnuEPaste_Click ()
    ' Send the clipboard text across the connection
    TNT1.Send = Clipboard.GetText()
End Sub
```

**Next: Resizing the VT Control**

### Resizing the VT control

It would be a nice feature if the VT control resized itself to the size of the window whenever the window was resized. To implement this, add the following code to the form's Resize event:

```
Sub Form_Resize ()
    ' Make the control fit the screen
    VT1.Left = 0
    VT1.Top = 0
    VT1.Height = Me.ScaleHeight
    VT1.Width = Me.ScaleWidth
End Sub
```

**Step 4: Run the program**

## Step 4: Run the program

The Telnet program is now complete. To test it, choose "Connect..." from the Session menu, type in the name of a Telnet server (most likely a UNIX host), and click Connect. You now have a fully-function Telnet application.



**Figure 3-4. The finished application.**

[Back to Contents](#)

## **About TCP**

The Transmission Control Protocol (TCP) is a data communications protocol - a set of rules for communications between computers. TCP provides a *reliable* stream of data between two applications. TCP is a general-purpose protocol with a large number of uses: TCP is the base protocol used for Telnet, FTP, and many other "upper-layer" protocols. Using TCP, you can implement any of these more complicated protocols.

[TCP Features](#)

[TCP/IP Connections](#)

[TCP and PowerTCP](#)

[More on TCP](#)

[TCP Sample Application](#)

[TCP Custom Control](#)

## TCP Features

### About TCP

TCP is built upon the IP protocol, which provides an *unreliable* transfer of data. Because TCP uses IP to transmit data, TCP must use methods to make it reliable. TCP provides the following features:

When one end of a TCP connection receives data, it sends a message back to the other end, acknowledging that it received it. If one end of a connection does not receive a confirmation after it sends data, it will re-send the data until it receives a confirmation or times out.

TCP keeps checksums on data sent. If the checksum on received data does not match the data sent, it will discard the data and not acknowledge having received the data. The other end will then re-send the data because it will not have received an acknowledgment.

Both ends of a TCP connection contain buffers to store data before it is used by applications. TCP makes sure that neither buffer will overflow. This is known as *flow control*.

Data submitted to TCP for transmission may be split up or combined into data segments which TCP considers the optimal size. Bytes sent from one end of a TCP connection will arrive at the other end in the same order, but not necessarily in the same size segments as they were submitted.

TCP ensures that bytes are ordered correctly. TCP implements this by checking the order of received data, deleting duplicate data if necessary, and acknowledging received packets.

TCP is a stream-oriented protocol. This means that the flow of data in TCP is without any predictable breaks in the data. It is up to an application to define meaningful record blocking.

TCP provides simultaneous two-way (full-duplex) data transfer. This means that data can flow from both ends of a connection at the same time.



## TCP Connections

### About TCP

TCP is a *connection-oriented* protocol. This means that a connection must be established between two computer systems before they can exchange data. The way this works is:

1. A client sends a message to a server requesting a connection.
2. The server responds, acknowledging the message to connect.
3. The client then sends a second message to the server acknowledging the server's message.

These three steps make the connection. The server must have already been listening for connections before the client could connect. The server made a *passive connection* (it was listening for client trying to connect). The client application which initiated the connection created an *active connection*.

Closing a connection works in a similar way. One end makes an *active close* (initiating a close). The other end then performs a *passive close*.

## TCP and PowerTCP

### About TCP

PowerTCP takes advantage of all the features that make TCP reliable, so the application does not have to be concerned about implementing flow control, checksums, or acknowledgments.

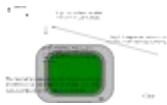
For developing client software that make active connections:

1. Connect to a remote computer by setting **Action = CONNECTCOMM**.
2. Send or process received data with the **Send** property and the **Recv** event.
3. Close the connection by setting **Action = CLOSECOMM**.

For developing server software that listen for connections:

1. Listen for incoming connections by setting **Action = LISTENCOMM**.
2. When a remote client tries to connect, respond to the **Accept** event by creating a new control and assigning the new connection to the new control using the **Session** property.
3. This new session can send or process received data by using the **Send** property and the **Recv** event.
4. Close the new session by setting **Action = CLOSECOMM**.
5. Stop listening by setting **Action = CLOSECOMM** when it is time to terminate the server.

Each new session is a new custom control. Figure 4-1 below demonstrates this operation:



**Figure 4-1. How a PowerTCP server accepts new connections**

This method allows the server to dynamically create a new session for each remote client trying to connect, which allows each session to be entirely independent from the listener. The listener can be subsequently shut down, or it can be left open to continue to accept passive connections. The number of passive connections generated is limited only by available memory and any limits imposed by the TCP/IP sub-system.

## More on TCP

### About TCP

One of the reasons TCP/IP has become so popular is because it has been implemented on so many operating systems and hosts. This is called *heterogeneous* operation, and is demonstrated whenever a PC is used to logon to an IBM host using Telnet, and then uses FTP to send a file to a VAX system half-way around the world. In figure 4-1 above we show how connections are established using PowerTCP, but we could substitute any TCP/IP host into the picture and still be completely accurate.

All Internet standards are published as *RFCs* (Request For Comment). These describe the network protocols, and other standards, in detail. To learn more about RFCs, see [Additional Resources](#).

## TCP Sample Application

### About TCP

The TCP sample application that comes with the PowerTCP Standard Toolkit for VB is an excellent example of how to use the TCP VBX. This sample demonstrates connectivity to some common ports available on many TCP/IP servers.

This section will explain the steps involved in constructing a simple application. The steps are to connect to a port, send some data, receive some data, repeat the two previous steps as often as necessary, and then close the connection.

1) Connect to a host. The CONNECT button, when selected, initiates a connection.

```
Sub ConnectButton_Click ()
    TCP1.RemotePort = 7           ' set the port to connect to
    TCP1.RemoteHost = HostName   ' set the hostname
    TCP1.Action = CONNECTCOMM   ' establish the connection
End Sub
```

2) Confirm the connection. After the connection is attempted, one of two things will happen: the Connect event will fire or the Exception event will fire.

The Connect Event Subroutine may include the following:

```
Sub TCP1_Connect ()
    Status = "Connected to " & TCP1.RemoteHost & " on port " & TCP1.RemotePort
    ' perform any UI update to indicate a connected state..
End Sub
```

If there was a problem, the Exception event will fire:

```
Sub TCP1_Exception (ErrorCode As Integer, ErrorDesc As String)
    Status = ErrorDesc ' show the user the Error Received
End Sub
```

3) Send Data. Once we are connected, data can be sent by selecting a button:

```
Sub GoButton_Click ()
    ' use the DataTag property to tag this send with a value
    TCP1.DataTag = Len(Data)
    ' now send some data from our Data edit box
    TCP1.Send = Data
End Sub
```

Once the Data is sent, the SendEvent will fire to tell us when the data has been accepted by the system buffers.

```
Sub TCP1_Send (DataTag As Long)
    BytesOut = Val(BytesOut) + DataTag ' update out cnt
End Sub
```

And, as always, if there is a problem, the Exception event will fire.

4) Receive Data. Data received by the application comes to it via the Recv event.

```
Sub TCP1_Recv (RecvData As String)
    If RecvData = "" Then
        ' Connection has been closed
        Status = "Closed"
    Else
        ' Append data into edit box
        RecvWindow.SelText = RecvData
    End If
End Sub
```

## 5) Close the Connection.

```
Sub CloseButton_Click ()  
    ' use CLOSECOMM to close connection gracefully  
    ' use ABORTCOMM to close connection abruptly  
    TCP1.Action = CLOSECOMM  
End Sub
```

The five steps outlined above illustrate how a simple TCP application is constructed. If you have any further questions, please refer directly to the sample source code.

## TCP Custom Control

[About TCP](#)



P16TCPB5.VBX

### Object Type

PowerTCP\_TCP

The properties and events for this control are listed in the following table. Properties and events that apply *only* to this control, or which require special consideration when used with it, are marked with an asterisk (\*). They are documented in the Properties and Events section.

### Properties

* <a href="#">Action</a>	* <a href="#">LocalName</a>	* <a href="#">Recv</a>	* <a href="#">Session</a>
* <a href="#">DataTag</a>	* <a href="#">LocalPort</a>	* <a href="#">RemoteHost</a>	* <a href="#">State</a>
* <a href="#">Flags</a>	Left	* <a href="#">RemotePort</a>	Tag
hWnd	Name	* <a href="#">Send</a>	Top
Index	* <a href="#">OemLicense</a>	* <a href="#">SendByte</a>	* <a href="#">Urgent</a>
* <a href="#">LocalDotAddr</a>	Parent	* <a href="#">SendString</a>	

### Events

* <a href="#">Accept</a>	* <a href="#">Exception</a>	* <a href="#">Recv</a>
* <a href="#">Connect</a>	* <a href="#">Listen</a>	* <a href="#">Send</a>

## About Telnet

Telnet is a communications protocol - a set of rules for communications between applications. Telnet stands for "Telecommunications Network Protocol." Telnet communication works between different types of computers and operating systems, because the protocol is the same for all systems. Telnet is often used for remote login (accessing one system from another over a network). A typical use for Telnet would be a terminal connecting with a server using Telnet to access the server's resources.

Many computers support the Telnet protocol. The main use for Telnet is for client-server based terminal communications. Telnet is used to log into remote systems and give commands. It can be used to access remote databases, control other systems, and distribute work between multiple systems, among other applications.

Telnet is similar to TCP - in fact, it is built upon it. But, unlike TCP, Telnet provides several additional features and concepts which makes it an "upper-layer" protocol.

Using Telnet, both ends of a connection send data to each other. Typically, a terminal will send commands to a server (or host), and the server will respond with messages or data. For example, a user at a terminal could type "help" and the server could respond with "Type a command to get help on".

Telnet provides *option negotiation* as a method for establishing conventions between two applications. During a session, either application may send the other a command, asking or telling about a specific option. A typical option would be "echoing". The process of option negotiation would determine if either application will support echoing. See the section "Option Negotiation" below for more information on option negotiation.

[Telnet Connections](#)

[The Network Virtual Terminal](#)

[Option Negotiation](#)

[Telnet and PowerTCP](#)

[More on Telnet](#)

[Telnet Sample Application](#)

[Telnet Custom Control](#)

## Telnet Connections

### About Telnet

Connections are made on a *port* - a method of distinguishing multiple sessions on a single computer. The standard Telnet port is 23. Most Telnet connections are made to port 23.

To make a connection, you must specify a port and also a host name or address to connect to. All TCP/IP systems on a network have an *IP address* - a number which distinguishes them from other systems. All IP addresses have the same format - *x.x.x.x* where each *x* is a number from 0 to 255 (except for the first *x*, which may be from 0 to 247). A typical IP address might be 129.229.1.2. Most networks also support host names, where the network will translate a name (such as "server3") into an IP address.

PowerTCP supports two types of connections - *active* and *passive*. An active connection is where an application tries to connect to a server. An example of this would be a terminal connecting to a Telnet server. A passive connection is where a computer receives a connection from a remote computer. An example of this would be a Telnet server waiting for terminal connections.



## The Network Virtual Terminal

### About Telnet

The Network Virtual Terminal (NVT) is a concept Telnet uses to represent both computers in a connection. Both ends of a Telnet connection must support the concept of an NVT. The NVT is an imaginary character-based device with a keyboard and a printer (input and output). Incoming data goes to the printer, and input from the keyboard goes to the other end of the Telnet connection.

The NVT supports the 7-bit character set known as *NVT ASCII*. NVT ASCII is the same as normal ASCII text, except that only 7 bits represent the character. The eighth high bit of the byte must be set to 0. In NVT ASCII, an end of line is represented by a two character sequence CR,LF (carriage return, linefeed). A carriage return is represented by the two character sequence CR,NULL (carriage return, null character).

Both ends of a Telnet connection map their own terminal device characteristics to and from the NVT. For example, a program must change all 8-bit ASCII characters to the NVT character set. The NVT is intended to strike a balance between being overly restrictive (not providing some computers a rich enough vocabulary for mapping into their local character sets), and being overly inclusive (penalizing users with modest terminals).

Please refer to RFC 854 for more information on the NVT.

## Option Negotiation

### About Telnet

Option negotiation is the process of two applications agreeing on what features they will or won't provide. Each application may send a command to the other, and each may respond to commands. A typical option negotiation session could proceed as shown below (although over 40 different options might be negotiated):

<i>Computer a</i>	<i>Computer b</i>
1. I will echo back what you send to me.	
	2. OK, do echo back what I send to you.
	3. Do suppress go-aheads.
4. No, I won't suppress go-aheads.	
	5. Don't suppress go-aheads.
6. OK, I won't suppress go-aheads.	

Either computer may initiate negotiation of an option. They may ask the other computer to support or not to support an option, or that they will or won't support an option. The other computer may then respond that they will or won't support an option, or ask the other computer to support or not to support an option. The four commands follow:

<b>Command</b>	<b>Description</b>
WILL	The sender wants to support the option
WONT	The sender will not support the option
DO	The sender wants the receiver to support the option
DONT	The sender wants the receiver to not support the option

The different pairs of commands and responding commands follow:

<b>Command</b>	<b>Response</b>
WILL I will support this option	DO OK, do support it
WILL I will support this option	DONT No, don't support it
WONT I won't support this option	DONT OK, don't support it
DO Do support this option	WILL OK, I will support it
DO Do support this option	WONT No, I won't support it
DONT Don't support this option	WONT OK, I won't support it

## NOTE

Notice that if a computer does not want an option supported (**WONT/DONT**), then the other computer must comply, responding with a **DONT** or **WONT**.

Telnet also provides *sub-option negotiation*. Sub-option negotiation means negotiating details about a specific option. An example would be the option "terminal type". For both computers to decide to support the same terminal type, you wouldn't want a different option for each individual terminal type, so you use a sub-option. A typical sub-option negotiation is shown below:

### Computer a

1. I will support different terminal types.

3. OK, send your terminal type sub-option.

### Computer b

2. OK, do support different terminal types.

4. My terminal type sub-option is "IBMPC".

As you can see, option negotiation and sub-option negotiation make Telnet very flexible and adaptable to many different terminal configurations.

Please refer to applicable RFCs for complete information on Telnet and Telnet option negotiation.

## Telnet and PowerTCP

### About Telnet

The additional functionality that Telnet provides above TCP is option negotiation. PowerTCP searches for all Telnet commands, strips them out of the data stream, and provides them to your application via a simple event notification. Automatic replies can also be enabled by using the AutoOption property. Convenient properties are also provided for sending out Telnet command strings.

### **TROUBLE**

Most difficulties have option negotiation as their cause. If your Telnet application "hangs" at any point, always check your option negotiation code first (or make sure that AutoOption = True if you want automatic option negotiation).

## **More on Telnet**

### About Telnet

All Internet standards are published as *RFCs* (Request For Comment). These describe the network protocols, and other standards, in detail. If you are interested in learning more about networking, see [Additional Resources](#).

### **NOTE**

The RFCs associated with Telnet are listed below (some of these have been included with your software distribution in RFCS.ZIP):

854, 855, 856, 857, 858, 859, 860, 861, 885, 927, 933, 946, 1041, 1043, 1053, 1073, 1079, 1080, 1091, 1096, 1097, 1112, 11122, 1143, 1184, 1205

## Telnet Sample Application

### About Telnet

The Telnet Sample application that comes with the PowerTCP Standard Toolkit for VB is an excellent example of how to use the Telnet VBX. This sample demonstrates connectivity to some common ports available on most Telnet servers.

This section will explain the steps involved in constructing a simple Telnet application. The steps are to connect to a port, send some keyboard input generated by the user, receive some data and display it, repeat the two previous steps as often as necessary, and then close the connection.

1) Connect to a Server. The following sample code is fired when the CONNECT button is selected:

```
Sub cmdConnect_Click ()
    ' Select from the possible service ports
    Select Case cmbPort
    Case "telnet"
        ' Standard telnet
        frmTelnet.TNT1.RemotePort = 23
    Case "echo"
        ' Echos back whatever is sent
        frmTelnet.TNT1.RemotePort = 7
    Case "discard"
        ' Does not echo back what is sent
        frmTelnet.TNT1.RemotePort = 9
    Case "daytime"
        ' Returns the date and time
        frmTelnet.TNT1.RemotePort = 13
    Case "chargen"
        ' Generates a stream of characters
        frmTelnet.TNT1.RemotePort = 19
    Case Else
        ' user put in another number
        frmTelnet.TNT1.RemotePort = Val(cmbPort)
    End Select

    frmTelnet.TNT1.RemoteHost = HostName      ' specify host
    frmTelnet.TNT1.Action = CONNECTCOMM ' initiate connection
    Unload Me
End Sub
```

2) Confirm the connection. After the connection is attempted, either the Connect event will fire or the Exception event will fire. The Connect Event Subroutine may include the following:

```
Sub TNT1_Connect ()
    ' Provide some UI feedback to user
    VT1.Cursor = True
    Me.Caption = " VT220 - " & Format$(TNT1.RemoteHost)
End Sub
```

3) Send Data. Once connected to a Telnet server, data is sent by typing on the keyboard. The VT control, which is used in the Telnet sample, has an event that fires when a key is pressed. The data in the KeyString is sent to the server by using the Send property.

```

Sub VT1_KeyPress (KeyString As String)
    ' Send the character to the host
    TNT1.Send = KeyString
    ' If the LocalEcho menu option is checked, then
    ' we should display it locally as well.
    If mnu0LocalEcho.Checked = True Then VT1.Display = KeyString
End Sub

```

**4) Receive Data.** Data received by this program come to it via the Recv event. This program moves the data to the VT220 display by assigning the RecvData parameter to the VT's Display property.

```

Sub TNT1_Recv (RecvData As String)
    If Len(RecvData) = 0 Then
        ' provide some UI feedback
        Me.Caption = " VT220"
        VT1.Cursor = False
    Else
        ' Place the received characters into the VT-220 emulator
        VT1.Display = RecvData
    End If
End Sub

```

**5) Close the Connection.** The Telnet session can be discontinued by simply executing the Disconnect function from the Menu. This function executes the CLOSECOMM action in order to destroy the Telnet Connection. Note that when complete, the Recv event will be fired to confirm this action.

```

Sub mnuCDisconnect_Click ()
    TNT1.Action = CLOSECOMM
End Sub

```

The five steps outlined above illustrate how a powerful Telnet Emulator can be constructed with only a few lines of code. If you have additional questions, please refer directly to the sample source code.

## Telnet Custom Control

[About Telnet](#)



P16TNTB5.VBX

### Object Type

PowerTCP\_TNT

The properties and events for this control are listed in the following table. Properties and events that apply *only* to this control, or which require special consideration when used with it, are marked with an asterisk (\*). They are documented in the Properties and Events section.

### Properties

<u>*Action</u>	Index	<u>*Recv</u>	<u>*SubOption</u>
<u>*Cmd</u>	<u>*LocalDotAddr</u>	<u>*RemoteHost</u>	Tag
<u>*DataTag</u>	<u>*LocalName</u>	<u>*RemotePort</u>	Top
<u>*DontOption</u>	<u>*LocalPort</u>	<u>*Send</u>	<u>*Urgent</u>
<u>*DoOption</u>	Left	<u>*SendByte</u>	<u>*WillOption</u>
<u>*DoSubOption</u>	Name	<u>*SendString</u>	<u>*WontOption</u>
<u>*Flags</u>	<u>*OemLicense</u>	<u>*Session</u>	
hWnd	Parent	<u>*State</u>	

### Events

<u>*Accept</u>	<u>*Connect</u>	<u>*Listen</u>	<u>*Send</u>
<u>*Cmd</u>	<u>*Exception</u>	<u>*Recv</u>	

## About VT Emulation

VT emulation is a terminal protocol. It differs from TCP and Telnet in that it does not include rules for data communications, but for data display. The VT protocols include VT52, VT100, and VT220. Each contains certain rules for interpreting incoming data. For example, one rule dictates that when a certain combination of characters is received, it means to clear the terminal screen.

Digital Equipment developed the VT emulation protocols for their VT terminals. This hardware contained a screen and keyboard. Eventually, the same VT emulation protocols were being used in terminal software programs for PCs. The VT52, 100, and 220 protocols became terminal emulation standards, along with others such as ANSI. This product implements these protocols and is available as a complementary part of the PowerTCP product.

For more information on VT emulation, contact Digital Equipment Corporation.

[VT Emulation and PowerTCP](#)

[Key Mapping](#)

[VT Character String Tables](#)

[VT Custom Control](#)



## VT Emulation and PowerTCP

### About VT Emulation

A VT emulation custom control is included with PowerTCP mainly to be used with the Telnet control. When data is received in the **Recv** event of the Telnet control, place the data into the VT control with a statement similar to the following:

```
Sub Telnet1_Recv (RecvData as String)
    VT1.Display = RecvData
End Sub
```

When the user types within the VT control, use a statement similar to the following to send it across the Telnet connection:

```
Sub VT1_KeyPress (KeyString As String)
    Telnet1.Send = KeyString
End Sub
```

In this way, data can be seamlessly transferred between the two controls.

## Key Mapping

### About VT Emulation

The custom control utilizes a default key mapping that is generally position-based. For example, the PF1 key on the VT220 is at the top of the keypad, so the NumLock key on the PC keyboard is mapped to the PF1 function. The following table summarizes the keyboard mappings:

VT Key	PC Key
Find	Insert
InsertHere	Home
Remove	Page Up
Select	Delete
PrevScreen	End
NextScreen	Page Down
PF1	Num Lock
PF2	/ (on numeric keypad)
PF3	* (on numeric keypad)
PF4	- (on numeric keypad)
, (comma)	+ (on numeric keypad)
- (minus)	Ctrl + "+" (numeric keypad)
Enter	Enter (on numeric keypad)
F6 - F10 (with Shift for user-defined keys)	F6 - F10 (with Ctrl for user defined keys)
F11 - F20 (with Shift for user-defined keys)	Shift + F1-Shift + F10 (with Ctrl for user defined keys)
Help	F11
Do	F12

There are two ways to modify these default mappings:

1. When the **KeyDown** Event is called, the KeyCode and Shift parameters can be changed to another value. For example, to remap F1 to PF1, when the F1 virtual key code is seen, it can be changed to the NumLock virtual key code. This will have the effect of generating the PF1 sequence whenever the F1 key is depressed (since F1 will generate the NumLock key code and the NumLock key code always generates a PF1).
2. When the **KeyDown** Event is called, the KeyCode parameter can be set to 0 (canceling out the default string normally generated by the key) and the user can send any arbitrary string directly out the communications channel. When F1 is pressed, for example, the user may program the "SS3 P" sequence.

## VT Character String Tables

[About VT Emulation](#)

The following tables are provided as a guide to the character strings generated by the VT220. The reader may wish to find more complete programming information directly from Digital Equipment Corporation.

### Control Sequences

Control Sequence	7-bit Mode	8-bit Mode
CSI	ESC [	0x9b
SS3	ESC O	0x8f

### Main Keypad Function Keys

VT/IBM Key	Code Transmitted
Backspace	BS character
Tab	HT character
Return	CR or CRLF depending upon the NewLine property
Ctrl, Lock, Shift	Does not send a code
Space Bar	SP character

### Edit Keys

VT Key	IBM Key	VT 220 Code	VT 100/52 Code
Find	Insert	CSI 1 ~	None
Insert Here	Home	CSI 2 ~	None
Remove	Page Up	CSI 3 ~	None
Select	Delete	CSI 4 ~	None
Prev Screen	End	CSI 5 ~	None
Next Screen	Page Down	CSI 6 ~	None

### Cursor Control Keys

VT/IBM Key	VT100/220 Code Normal Cursor Key Mode	VT100/220 Code Application Cursor Key Mode	VT52 Code Normal Cursor Key Mode	VT52 Code Application Cursor Key Mode
Up Arrow	CSI A	SS3 A	ESC A	ESC A
Down Arrow	CSI B	SS3 B	ESC B	ESC B
Right Arrow	CSI C	SS3 C	ESC C	ESC C
Left Arrow	CSI D	SS3 D	ESC D	ESC D

### Auxiliary Keypad Keys

VT Key	IBM Key	VT100/20 Code Numeric Keypad Mode	VT100/220 Code Application Keypad Mode	VT52 Code Numeric Keypad Mode	VT52 Code Application Keypad Mode
--------	---------	--	---	--	--

0	0	0	SS3 p	0	ESC ? p
1	1	1	SS3 q	1	ESC ? q
2	2	2	SS3 r	2	ESC ? r
3	3	3	SS3 s	3	ESC ? s
4	4	4	SS3 t	4	ESC ? t
5	5	5	SS3 u	5	ESC ? u
6	6	6	SS3 v	6	ESC ? v
7	7	7	SS3 w	7	ESC ? w
8	8	8	SS3 x	8	ESC ? x
9	9	9	SS3 y	9	ESC ? y
-	Ctrl + "+"	- (minus)	SS3 m	-	ESC ? m
,	+	, (comma)	SS3 l	,	ESC ? l
.	.	. (period)	SS3 n	.	ESC ? n
Enter	Enter	CR or CR/LF	SS3 M	CR or CR/LF	ESC ? M
PF1	NumLock	SS3 P	SS3 P	ESC P	ESC P
PF2	/	SS3 Q	SS3 Q	ESC Q	ESC Q
PF3	*	SS3 R	SS3 R	ESC R	ESC R
PF4	-	SS3 S	SS3 S	ESC S	ESC S

## Top Row Function Keys

VT Key	IBM Key	VT220 Code	VT100/52 Code
Hold Screen	F1	(none)	(none)
Print Screen	F2	(none)	(none)
Set-up	F3	(none)	(none)
Data/Talk	F4	(none)	(none)
Break	F5	(none)	(none)
F6	F6	CSI 1 7 ~	(none)
F7	F7	CSI 1 8 ~	(none)
F8	F8	CSI 1 9 ~	(none)
F9	F9	CSI 2 0 ~	(none)
F10	F10	CSI 2 1 ~	(none)
F11 (ESC)	<shift> F1	CSI 2 3 ~	ESC
F12 (BS)	<shift> F2	CSI 2 4 ~	BS
F13 (LF)	<shift> F3	CSI 2 5 ~	LF
F14	<shift> F4	CSI 2 6 ~	(none)
Help	F11	CSI 2 8 ~	(none)
Do	F12	CSI 2 9 ~	(none)
F17	<shift> F7	CSI 3 1 ~	(none)
F18	<shift> F8	CSI 3 2 ~	(none)
F19	<shift> F9	CSI 3 3 ~	(none)
F20	<shift> F10	CSI 3 4 ~	(none)

## VT Custom Control

[About VT Emulation](#)



P16VT2B5.VBX

## Object Type

PowerTCP\_VT2

The properties and events for this control are listed in the following table. Properties and events that apply *only* to this control, or which require special consideration when used with it, are marked with an asterisk (\*). They are documented in the following sections.

### Properties

<u>*AnswerBack</u>	<u>*CursorRow</u>	Index	<u>*Rows</u>
<u>*AutoPrint</u>	<u>*CursorStyle</u>	<u>*Keypad</u>	<u>*Scroll</u>
<u>*AutoRepeat</u>	<u>*Display</u>	Left	<u>*SelLength</u>
<u>*AutoWrap</u>	<u>*DisplayString</u>	MousePointer	<u>*SelStart</u>
<u>*BackColor</u>	<u>*Enabled</u>	Name	<u>*Style</u>
<u>*Bell</u>	<u>*EnableNewLine</u>	<u>*NewLine</u>	<u>*Terminal</u>
<u>*BoldColor</u>	FontBold	<u>*OemLicense</u>	<u>*Text</u>
<u>*BufferRows</u>	FontItalic	Parent	TabIndex
<u>*Cols</u>	<u>*FontName</u>	<u>*PrinterControlle</u>	<u>*Tabs</u>
<u>*ColWidth</u>	FontSize	<u>l</u>	TabStop
<u>*Cursor</u>	<u>*ForeColor</u>	<u>*PrintPassthroug</u>	Tag
<u>*CursorCol</u>	Height	<u>h</u>	Top
<u>*CursorKeys</u>	hWnd	<u>*PrintScreen</u>	Width
		<u>*Reset</u>	
		<u>*RowHeight</u>	

### Events

<u>*Click</u>	<u>*KeyDown</u>	<u>*NewLine</u>
<u>*HostCommand</u>	<u>*KeyPress</u>	

### Methods

\*Clear

## **About FTP**

FTP stands for File Transfer Protocol. It provides a standard method for transferring files between computers. Using FTP, you can get directory listings, retrieve files, upload files, and more. An FTP client accesses files stored on a server. An FTP server responds to FTP clients, providing directory and file data. FTP is based upon both TCP and Telnet, so that FTP differs from other protocols in that it actually consists of *two* connections - a control connection which uses Telnet, and a data connection which uses TCP.

[FTP Connections](#)

[Data Representation](#)

[Common FTP Commands](#)

[FTP and PowerTCP](#)

[FTP Sample Application](#)

[FTP Custom Control](#)

## FTP Connections

### About FTP

When an FTP session is initiated, a control connection is created. The control connection is used for login, sending commands, and receiving information. The control connection utilizes the Telnet protocol. It usually uses the standard FTP port 21.

An FTP connection is generally made between a client and server. However, a client can control file transfers between two servers. See the FTP RFC (number 959) for detailed information about this and other FTP features.

During login, a user name and password must be given. Many FTP servers support *anonymous FTP*, where the user name is "anonymous" and the password is the user's email address.

Once the login is complete, commands can be sent across the control connection. Typical commands are RETR (retrieve a file), STOR (upload a file), and QUIT (logoff). For more information, please refer to the RFC.

An FTP server responds to commands with three-digit codes and descriptions across the control connection. Each of the three digits have a different meaning, shown in the chart below. The code descriptions are taken directly from the RFC.

### First-digit codes:

Code	Description
1yz	Positive Preliminary reply. The requested action is being initiated; expect another reply before proceeding with a new command. (The user-process sending another command before the completion reply would be in violation of protocol; but server-FTP processes should queue any commands that arrive while a preceding command is in progress.) This type of reply can be used to indicate that the command was accepted and the user-process may now pay attention to the data connections, for implementations where simultaneous monitoring is difficult. The server-FTP process may send at most one 1yz reply per command.
2yz	Positive Completion reply. The requested action has been successfully completed. A new request may be initiated.
3yz	Positive Intermediate reply. The command has been accepted, but the requested action is being held in abeyance, pending receipt of further information. The user should send another command specifying this information. This reply is used in command sequence groups.
4yz	Transient Negative Completion reply. The command was not accepted and the requested action did not take place, but the error condition is temporary and the action may be requested again. The user should return to the beginning of the command sequence, if any. It is difficult to assign a meaning to "transient", particularly when two distinct sites (Server- and User-processes) have to agree on the interpretation. Each reply in the 4yz category might have a slightly different time value, but the intent is that the user-process is encouraged to try again. A rule of thumb in determining if a reply fits into the 4yz or the 5yz (Permanent Negative) category is that replies are 4yz if the commands can be repeated without any change in command form or in properties of the User or Server (e.g., the command is spelled the same with the same arguments used; the user does not change his file access or user name; the server does not put up a new implementation.)

5yz Permanent Negative Completion reply. The command was not accepted and the requested action did not take place. The User-process is discouraged from repeating the exact request (in the same sequence). Even some "permanent" error conditions can be corrected, so the human user may want to direct his User-process to reinitiate the command sequence by direct action at some point in the future (e.g., after the spelling has been changed, or the user has altered his directory status.)

## Second-digit codes

(x4z has not yet been defined by the RFC):

Code	Description
x0z	Syntax. These replies refer to syntax errors, syntactically correct commands that don't fit any functional category, and unimplemented or superfluous commands.
x1z	Information. These are replies to requests for information, such as status or help.
x2z	Connections. Replies referring to the control and data connections.
x3z	Authentication and accounting. Replies for the login process and accounting procedures.
x5z	File system. These replies indicate the status of the Server file system vis-à-vis the requested transfer or other file system action.

The third-digit codes are more detailed. Refer to the RFC for these. Normally, an FTP response will consist of the code followed by an English description. Examples:

```
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
226 Transfer complete.
221 Goodbye.
```

A data connection is opened when one of three things happen:

- The client sends a file to the server
- The server sends a file to the client
- The server sends a directory listing to the client

Once a data connection has transferred its data, it is closed. The control connection always exists, but the data connection is dynamic. Note that only one data connection can exist at the same time. The following steps are taken to create the data connection:

1. The client chooses an unused port number to accept the connection on, because the client is always in control of file transfers.
2. The client waits for a connection on that port.
3. The client sends the port number to the server over the control connection, using the PORT command.
4. The client sends a command which transfers data across a data connection (such as STOR, RETR, or LIST).
5. The server connects to the correct port using its local port 20.

Once the data connection is established, a file transfer or directory listing can take place. The connection is closed after the transfer takes place.



## Data Representation

### About FTP

Data can be transferred over the data connection in many different modes. File type, format control, structure, and transmission mode can all be controlled. However, the only commonly used feature is file type. The rest will generally not be changed.

There are four options for file type, but ASCII and IMAGE are most commonly used:

- **ASCII.** This is the default mode. Different computer systems implement new-line characters in different ways. Many UNIX systems use the ASCII character 13 (CR). Microsoft Windows uses ASCII character 13 followed by 10 (CR-LF). When ASCII mode is enabled, proper conversions are made between the server and client. This mode should be used only for text files. Using this mode on binary files will cause them to become corrupted.
- **EBCDIC.** This is not commonly used. It is an alternative way of transferring files between two EBCDIC systems.
- **IMAGE (binary).** This sends an exact replica of the file across the connection, preserving each byte with no conversions. It should be used for all binary files, which include all programs and many document files.
- **Local file type.** This is not commonly used. It is a method of transferring data between two computers with different byte sizes. However, virtually all computers today use 8-bit bytes, so this is not applicable.

PowerTCP ensures the proper data type is set before any data transfer is initiated.

## Common FTP Commands

### About FTP

The table below lists some of the more common commands used in FTP. The RFC contains all valid commands.

Command	Description
ABOR	Cancels the previous command issued, and halts any data transfer in progress.
LIST <i>filelist</i>	Lists all files and directories in the directory.
PASS <i>password</i>	Sends the user's password to the server during login.
PORT <i>n1,n2,n3,n4,n5,n6</i>	Sends the client's IP address in the first four arguments and the port in the fifth and sixth arguments.
QUIT	Logs off the server.
RETR <i>filename</i>	Retrieves a file from the server.
STOR <i>filename</i>	Places a file on the server.
TYPE <i>type</i>	Specifies the file type (A for ASCII, or I for IMAGE)
USER <i>username</i>	Sends the user's name to the server.

## FTP and PowerTCP

### About FTP

- 1. Login.** PowerTCP combines the control connection establishment process with the login (authentication) process. When the LoginHost property is set, the FTP library resolves the host name, makes the connection, handles the entire authentication process, and notifies the application when it has completed.
- 2. Data Connection Establishment.** PowerTCP handles all the details of ensuring the proper data representation type is set, and automatically establishing a data connection (listening on a local port, sending the PORT command, accepting a passive connection, etc.) when any of the data transfer commands are used.
- 3. File spooling.** If a local file is specified, PowerTCP will handle transferring data to and from that file automatically. However, the program can still opt to have complete control over all data transferred.
- 4. Renaming Files.** PowerTCP sequences the RNFR and RNT0 commands so the application does not have to.
- 5. Restart.** PowerTCP sequences the REST command with the data transfer command currently in progress.
- 6. Error Checking and Synchronization.** PowerTCP saves state information which is necessary for the correct interpretation of replies coming from the server. Replies are then reported to the application with the associated status information. All replies are checked for validity against the last command sent.
- 7. Complete RFC 959 functionality.** If you need the flexibility of using your own command strings, this is supported, bypassing any status-checking. PowerTCP can even be used to control two FTP servers, transferring files between them!

A property is provided for each of the FTP commands (except where related commands have been sequenced together for your convenience). For example, single commands like HELP

and PWD have one property associated with each. Other command sequences, such as USER, PASS and ACCT are fired off by a single property, where PowerTCP takes care of the sequencing. File transfer properties (STOR, STOU, APPE) automatically sequence TYPE, PORT and transfer commands, greatly simplifying the mechanism of completing passive data port connections.

It may still be necessary, however, for the application to know what codes are expected back when a command is sent. Although PowerTCP encapsulates many of the details, and a "Status" enumerated type simplifies this task, you may wish to recognize the successful completion of any command.

In general, there is a one-to-one correspondence between FTP commands and PowerTCP library functions. Exceptions are Store, Retrieve, Append, StoreUnique, LoginHost, Rename, and Restart, which combine several commands into a single property. The libraries "remember" what command was sent last, checking for valid and invalid replies received back from the FTP server.

The documentation that follows later in this help file describes each property. Please refer to RFC 959 for a thorough description of each command, along with more general descriptions and guidance describing how the FTP protocol operates.

## FTP Sample Application

### About FTP

The FTP Sample application that comes with the PowerTCP Standard Toolkit for VB is an excellent example of how to use the FTP VBX. This sample demonstrates connectivity to an FTP Server and implements many of the features available in the FTP Protocol.

This section will explain the steps involved in constructing a simple application. The steps are to connect to an FTP server, execute commands defined by the RFC and then Logout.

1) Connect to an FTP Server. The LOGIN button, when selected, initiates the following code. The LoginHost Property actually triggers the connection attempt.

```
Sub Command1_Click ()
FTP1.User = User
    FTP1.Password = Pass
    FTP1.Account = Acct
    FTP1.LoginHost = Host ' initiate a connection
End Sub
```

2) Confirm the connection. After the connection is attempted, the Connect event will fire or the Exception event will fire. If the Connect event fires, then the user will receive replies from the FTP server via the Reply event. You will only need to provide code for the Connect event if you wish to capture the parameters provided there. The connection can be confirmed the first time the Reply event is fired as shown below.

```
Sub FTP1_Reply (Status As Integer, LastCommand As Integer,
                Code As Integer, Reply As String)
If Reply <> "" Then Text1.SelText = Reply ' show the reply
Select Case (Status)
Case FTP_UNKNOWN
    ' spontaneous data or reply from Command
    Status1 = "Unknown"
Case FTP_SUCCESS
    ' operation completed successfully
    ' if LastCommand=FTP_PASS, we know we're logged in
    ' if LastCommand=FTP_RETR, we know we have the file now
    ' if LastCommand=FTP_STOR, we know the file has gone to the host
    ' if LastCommand=FTP_LIST, we have received the directory listing
    ' if LastCommand=FTP_CLOSED, control connection is closed
    ' etc . . .
    Status1 = "Success"
    ' launch you next request, like next file to transfer
Case FTP_ERROR
    ' unexpected error
    Status1 = "Error"
Case FTP_FAILURE
    ' failure to complete successfully
    Status1 = "Failure"
Case FTP_WORKING
    ' informative...wait for next reply
    Status1 = "Working"
End Select
End Sub
```

3) Executing FTP Functionality. Once connected, the FTP program can now execute any of the FTP actions that are listed in the drop down list box by simply selecting one of them and then selecting the **Send Command** button.

```
Sub Command18_Click ()
If Auto Then
    ' indicates PowerTCP will auto-spool data
```

```

        ' LocalFileSpec property keys auto-spooling
        FTP1.LocalFileSpec = LocalName
    Else
        ' indicates your program will open and spool file
        FTP1.LocalFileSpec = ""
    End If
    ' specify how much data will read/written for each file access
    FTP1.BufferSize = BufferSize
    Select Case Comb1.Text
    Case "APPE"
        FTP1.Appe = PathName
    Case "CDUP"
        FTP1.ChDirUp = True
    Case "CWD"
        FTP1.ChDir = PathName
    Case "DELE"
        FTP1.Dele = PathName
    Case "LIST"
        FTP1.List = PathName
    Case "MKD"
        FTP1.MakeDir = PathName
    Case "NLST"
        FTP1.NameList = PathName
    Case "PWD"
        FTP1.PrintWorkingDir = True
    Case "RENAME"
        FTP1.Rename = PathName
    Case "RETR"
        FTP1.Retrieve = PathName
    Case "RMD"
        FTP1.RemoveDir = PathName
    Case "STOR"
        FTP1.Store = PathName
    Case "STOU"
        FTP1.StoreUnique = PathName
    End Select
End Sub

```

4) Receive File and Listing Data. If LocalFileSpec is set to "", then all Data received by this application comes to it via the Recv event or. If LocalFileSpec is set to a value, then the Transfer event is fired to notify your program of the transfer's progress. This is also the mechanism for receiving directory listings.

On a retrieve with the LocalFileSpec set to NULL, the file 'pieces' will arrive through this event:

```

Sub FTP1_Recv (RecvData As String)
    ' put data into open file
    If RecvData = "" Then
        ' close file that was previously opened...FTP server is done
        Close #1
        FileNum = 0
    Else
        ' put data into file, but we could do anything with it
        ' if a listing, you could display it instead. . .
        Print #1, RecvData;
    End If
End Sub

Sub FTP1_Transfer (LastCommand As Integer, BlockCnt As Long, ByteCnt As Long)
    ' put up some kind of UI progress indicator...
End Sub

```

5) Close the Connection. The control connection can be closed at any time by simply executing the Logout function as is done in the LOGOUT button on the form.

```
Sub Command2_Click ()  
    FTP1.Logout = True  
    ' Reply event will be fired with LastCommand=FTP_QUIT and  
    ' again with LastCommand=FTP_CLOSED  
End Sub
```

The five steps outlined above illustrate how the FTP control is used. If you have further questions, please refer directly to the sample source code.

## FTP Custom Control

[About FTP](#)



P16FTPB5.VBX

### Object Type

PowerTCP\_FTP

The properties and events for this control are listed in the following table. Properties and events that apply *only* to this control, or which require special consideration when used with it, are marked with an asterisk (\*). They are documented in the following sections.

### Properties

<u>*Abort</u>	hWnd	<u>*Noop</u>	<u>*SendString</u>
<u>*Account</u>	Index	<u>*OemLicense</u>	<u>*Site</u>
<u>*Allocate</u>	<u>*LastCommand</u>	Parent	<u>*Status</u>
<u>*Appe</u>	Left	<u>*Passive</u>	<u>*Store</u>
<u>*BufferSize</u>	<u>*List</u>	<u>*Password</u>	<u>*StoreUnique</u>
<u>*ChDir</u>	<u>*ListenTimeout</u>	<u>*Port</u>	<u>*StructMount</u>
<u>*ChDirUp</u>	<u>*LocalFileSpec</u>	<u>*PrintWorkingDir</u>	<u>*System</u>
<u>*CloseControl</u>	<u>*LocalDotAddr</u>	<u>*Reinitialize</u>	Tag
<u>*CloseData</u>	<u>*LoginHost</u>	<u>*RemoveDir</u>	Top
<u>*Command</u>	<u>*LogOut</u>	<u>*Rename</u>	<u>*Type</u>
<u>*Dele</u>	<u>*MakeDir</u>	<u>*Restart</u>	<u>*TypeTransfer</u>
<u>*FileStruct</u>	<u>*Mode</u>	<u>*Retrieve</u>	<u>*User</u>
<u>*Flags</u>	Name	<u>*Send</u>	
<u>*Help</u>	<u>*NameList</u>	<u>*SendByte</u>	

### Events

<u>*Connect</u>	<u>*Log</u>	<u>*Reply</u>	<u>*Transfer</u>
<u>*Exception</u>	<u>*Recv</u>	<u>*Send</u>	

## **About SMTP**

SMTP is a protocol used for sending electronic mail. SMTP is built upon TCP, like many other protocols. Note that SMTP is used only for *sending* mail. Receiving mail is handled by a separate protocol, such as POP3.

PowerTCP also supports automatic UUENCODE and MIME encoding for attachments as desired.

[Features of SMTP](#)

[SMTP and PowerTCP](#)

[SMTP Sample Application](#)

[SMTP Constants](#)

[SMTP Custom Control](#)



## SMTP Commands

### About SMTP

The eight most commonly used SMTP commands are listed below:

Command	Description
DATA	Used to send the body of a mail message.
HELO	Used to inform the server of the sending computer's name.
MAIL	Used to inform the server of the originator of the mail message.
NOOP	Causes the server to respond with an OK. NOOP stands for NO OPeration.
RCPT	Used to inform the server of the recipient(s) of the mail message.
RSET	Aborts any current mail transfer and resets both ends of the connection.
QUIT	Used to send the mail and close the connection.
VERFY	Sent from the client to the sender for verification of a recipient address.

A piece of electronic mail consists of three parts:

- **The envelope.** This contains information about the originator and recipients. See RFC 821 for more information.
- **The headers.** These are lines at the beginning of the mail message which contain other information. They consist of a title, colon, and data. For example:

Subject: This is a sample header

- **The body.** This contains the actual text of the message.

SMTP uses NVT ASCII for transmitting all mail messages (see the Telnet section for information on NVT ASCII).

## SMTP and PowerTCP

### About SMTP

The PowerTCP SMTP custom control encapsulates SMTP client operation as completely as possible, using a high-level interface for many programming environments. The SMTP custom control is based upon the PowerTCP SMTP C++ Class.

PowerTCP complies with RFC 821, Simple Mail Transfer Protocol client operations, handling most of the details of the protocol. You may wish to familiarize yourself with this document which is included with the PowerTCP Toolkit. Properties are provided for initiating SMTP activity, and events are generated to provide your program with data and/or notification that additional data can be sent. In accordance with RFC 821, the following functionality is provided:

- **Mail.** PowerTCP handles all sequencing of messages to send mail to a host, and optionally builds header information into the message body.
- **Verifying and Expanding.** Verification of mail addresses and expanding of mailing lists is supported.
- **Sending and Mailing.** Send, Send or Mail, Send and Mail are supported.
- **Hello and Quit.** HELO and QUIT messages are automatically used for channel opening and closing.
- **Relaying.** Specify a forward path as a parameter to utilize relaying facilities.

To utilize the PowerTCP SMTP custom control, 3 basic steps are involved:

1. Your application connects to a mail server. Upon successful completion, PowerTCP sends a HELO message to the host.
2. Your application can now use the Mail, Help, Verify, and Expand properties to accomplish desired activity.
3. Your application uses the Action property to send the QUIT message to terminate the session.

Each PowerTCP component handles a single TCP channel to the host. The following table shows what action properties cause what SMTP replies to be sent by the SMTP server:

PowerTCP API Call	SMTP Log (PowerTCP sends)	Reply from SMTP Server	Smtp Event LastCommand Status
Action = CONNECTCOMM	'Connect'	220 test Sendmail -TestMail Inc ready at Mon, 8 Jan 1996 12:08:54 -0500 gripes to root@test.domain.com	SMTP_CONNECT SMTP_WORKING
	HELO test	250 test.domain.com Hello me (me.domain.com), pleased to meet you.	SMTP_HELO SMTP_SUCCESS
Action = SEND_MAIL	MAIL FROM: test	250 test... Sender ok	SMTP_MAIL SMTP_WORKING
	RCPT TO: test	250 test... Recipient ok	SMTP_RCPT SMTP_WORKING
	DATA	354 Enter mail, end with "." on a line by itself	SMTP_DATA SMTP_WORKING
	Test Message !!!	250 Ok	SMTP_DATA SMTP_SUCCESS
Action = CLOSECOMM	QUIT	221 test.domain.com closing connection	SMTP_QUIT SMTP_CLOSED



## SMTP Sample Application

### About SMTP

The SMTP sample application that comes with the PowerTCP Standard Toolkit for VB is an excellent example of how to use the SMTP VBX.

The steps involved to send mail are as follows: Connect to a SMTP Server, Mail the message, and then close the connection.

1) Connect to an SMTP Server. The SEND button, when selected, initiates a connection.

```
Sub Command2_Click ()
    ' Set required parameters in order to perform a successful MAIL
    ' RemoteHost is the mail server that will accept the mail
    SMTP1.RemoteHost = HostName
    ' Set ACTION to make an attempt to connect to the server
    SMTP1.Action = CONNECTCOMM
End Sub
```

2) If the Connect event is fired, then the Smtp event will also fire with received information. From the information, we can determine what to do next. If the Smtp event is fired with LastCommand=SMTP\_HELO and Status=SMTP\_SUCCESS, then the SMTP Server is ready for our mail.

When all of the mail has been sent, the Smtp event will fire with a LastCommand of SMTP\_DATA and Status of SMTP\_SUCCESS. At that point it is safe to send another message, or close the connection. In the sample, we choose to close the connection.

```

Sub SMTP1_Smtp (Status As Integer, LastCommand As Integer, ReplyCode As Integer,
ReplyStr As String, Complete As Integer)

    ' display the reply from the mail server
    Text5.SelText = ReplyStr

    If LastCommand = SMTP_DATA Then
        ' mail has been sent...close down connection
        SMTP1.Action = CLOSECOMM 'close
        Text5.SelText = "Connection Closed !!" & Chr$(13) & Chr$(10)
        Exit Sub
    End If

    ' If the SMTP_SUCCESS (Good status from the Smtplib Server)
    ' and a good connection has been
    ' established (LastCommand=SMTP_HELO)
    ' then go ahead and send the message.
    If Status=SMTP_SUCCESS And LastCommand = SMTP_HELO Then
        ' Set your address for the header
        SMTP1.Sender = UserName
        ' list of all recipients
        SMTP1.Recipients = Destination
        ' PowerTCP will construct the header
        SMTP1.HeaderSubject = Subject
        ' set the message to be sent
        SMTP1.Message = Message.Text
        ' Set up the Attachments and their types
        ' we construct comma-delimited attachments with encoding mode
        SMTP1.Attachments = ""
        SMTP1.AttachTypes = ""
        For i = 0 To List1.ListCount - 1
            SMTP1.Attachments = SMTP1.Attachments & List1.List(i) & ","
            SMTP1.AttachTypes = SMTP1.AttachTypes & Left$(Attachtype.Text, 1) & ","
        Next i

        If SMTP1.Attachments <> "" Then
            ' clear the extra comma we put in
            SMTP1.Attachments = Left$(SMTP1.Attachments, Len(SMTP1.Attachments) - 1)
            SMTP1.AttachTypes = Left$(SMTP1.AttachTypes, Len(SMTP1.AttachTypes) - 1)
        End If

        SMTP1.Action = SEND_MAIL ' send the mail message!
    End If

End Sub

```

3) If the Mail message has attachments, then the Attach event will fire each time a buffer of data is sent to the SMTP server. The Attach event will report PercentComplete of the entire mail message transferred so the programmer can display progress information. The sample application displays a message box with a gauge that displays the percent transferred.

```

Sub SMTP1_Attach (FileSpec As String, PercentComplete As Long)
    If PercentComplete = 100 Then ' the file transfer is done
        Form3.Hide
    End If
    If PercentComplete = 0 Then ' file transfer is beginning
        Form3.Show
    End If
    If PercentComplete > 0 And PercentComplete < 100 Then
        Form3.Caption = FileSpec
        Form3.Text2.Width = PercentComplete * 60
    End If
End Sub

```



## SMTP Constants

### About SMTP

The following constants should be used to control and monitor SMTP activity.

#### ' Status in the Smtplib event

```
Global Const SMTP_CLOSED = 0 ' connection is closed
Global Const SMTP_SUCCESS = 1 ' successful reply
Global Const SMTP_ERROR = 2 ' error reply
Global Const SMTP_FAILURE = 3 ' failure reply
Global Const SMTP_WORKING = 4 ' PowerTCP working
```

#### ' LastCommand in the Smtplib event

```
Global Const SMTP_DATA = 0 ' Data (Mail) being sent
Global Const SMTP_EXPN = 1 ' Expand was performed
Global Const SMTP_HELO = 2 ' Hello greeting was done.
Global Const SMTP_HELP = 3 ' Help was requested
Global Const SMTP_MAIL = 4 ' MAIL command was sent
Global Const SMTP_NOOP = 5 ' NOACTION command
Global Const SMTP_QUIT = 6 ' QUIT Command
Global Const SMTP_RCPT = 7 ' Recipient Command sent
Global Const SMTP_RSET = 8 ' RESET Command
Global Const SMTP_SAML = 9 ' SEND and MAIL Command
Global Const SMTP_SEND = 10 ' SEND Command
Global Const SMTP_SOML = 11 ' SEND or MAIL Command
Global Const SMTP_TURN = 12 ' NOT IMPLEMENTED
Global Const SMTP_VRFY = 13 ' VERIFY Command
Global Const SMTP_CONNECT = 14 ' first greeting received from host
```

#### ' Actions for SMTP

```
Global Const CONNECTCOMM = 0 ' Connect to host
Global Const CLOSECOMM = 2 ' Close connection
Global Const ABORTCOMM = 3 ' Abort connection
Global Const RESET_MAIL = 4 ' Reset server
Global Const SEND_MAIL = 5 ' Send mail (normally used)
Global Const SEND_SEND = 6 ' Send
Global Const SEND_SOML = 7 ' Send or mail
Global Const SEND_SAML = 8 ' Send and mail
Global Const NOOP_MAIL = 11 ' No operation
```

## SMTP Custom Control

[About SMTP](#)



P16SMTP5.VBX

### Object Type

PowerTCP\_SSMTP

The properties and events for this control are listed in the following table. Properties and events that apply *only* to this control, or which require special consideration when used with it, are marked with an asterisk (\*). They are documented in the Properties and Events section.

### Properties

<u>*Action</u>	<u>*HeaderTo</u>	<u>*Message</u>	<u>*State</u>
<u>*Attachments</u>	<u>*Help</u>	Name	Tag
<u>*AttachTypes</u>	hWnd	<u>*OemLicense</u>	Top
<u>*Expand</u>	Index	Parent	<u>*Verify</u>
<u>*Flags</u>	Left	<u>*Recipients</u>	
<u>*HeaderDate</u>	<u>*LocalDotAddr</u>	<u>*RemoteHost</u>	
<u>*HeaderFrom</u>	<u>*LocalName</u>	<u>*RemotePort</u>	
<u>*HeaderSubject</u>	<u>*LocalPort</u>	<u>*Sender</u>	

### Events

<u>*Attach</u>	<u>*Exception</u>	<u>*Smtg</u>
<u>*Connect</u>	<u>*Log</u>	

## About POP3

The objective of the Post Office Protocol is to allow mail to be received on a remote host (the POP3 server) that is always available to accept mail on your behalf, and to make that mail available to your system (the POP3 client) upon demand. This allows the flexibility to have a PC intermittently connected to the Internet, yet still have local mail management. The PowerTCP POP3 control encapsulates POP3 client operation as completely as possible, using a high-level interface for many programming environments.

The SMTP control documented in the previous chapter is often used in conjunction with the POP3 control to provide a complete e-mail solution in a PC operating environment.

[POP3 and PowerTCP](#)

[POP3 Sample Application](#)

[POP3 Constants](#)

[POP3 Custom Control](#)



## POP3 and PowerTCP

### About POP3

PowerTCP complies with RFC 1725, Post Office Protocol Version 3, handling most of the details of the protocol. MIME and UUENCODE decoding of file attachments is also supported. You may wish to familiarize yourself with this document which is included with the PowerTCP Toolkit. Methods are provided for initiating POP3 activity, and events are generated to provide your program with data and/or notifications. In accordance with RFC 1725, the following functionality is provided:

Channel Initialization and Authorization. PowerTCP optimizes the TCP connection process and integrates it with authorization process. A single method is called to automate the connection and authorization process.

POP3 messaging. Methods are provided for most POP3 protocol commands: STAT, LIST, RETR, DELE, NOOP, RESET, TOP, and UIDL. (APOP, USER and PASS are integrated into the connect process, and QUIT is integrated into the close process.)

Receiving mail. An event is fired for receiving full or partial (top) mail messages.

Receiving lists. An event is fired when scan listings and drop listings are received.

Automatic UUENCODE and MIME. UUENCODE and MIME algorithms are invoked automatically when sensed in received mail. Attachments are automatically spooled to disk.

To utilize the PowerPOP3 Class Library, 3 basic steps are involved:

1. Connect to a POP3 server using the **Action=CONNECTCOMM**. PowerTCP uses the APOP command if the *Secret* parameter is specified, otherwise the *UserName* and *Password* parameters are used. If the APOP command fails, PowerTCP attempts to use the *UserName* and *Password* parameters.
2. Use the **DeleteMsg, List, Retrieve, Status, Top, TopLines, and Uidl** properties as required by your application.
3. Close the connection using the **Close ( FALSE )** method and wait for a final **OnPop3** to be called before terminating your application.

Note that each POP3 object handles a single TCP channel to a POP3 server.

The table below illustrates a sample POP3 session with the typical replies from the server and corresponding PowerTCP event calls. The first column shows the API call and in braces the corresponding POP3 command(s) generated on your behalf. Note that the second Pop3Event() generated in response to a Top() command provides a status of POP3\_MAIL and can include header and message text if more than 0 lines of the mail are requested. The second Pop3Event() generated in response to a Retrieve() will have a status of POP\_HEADER and contain the complete header, and subsequent Pop3Event() calls will have a status of POP3\_MAIL until the message is complete.

The FileEvent() and Pop3Event() related to processing attachment data provide the filename of the spooled file. Note that if Flags = PT\_OVERWRITE in the Connect() call, this filename may not match the name provided in the attachment header, as in the example which follows. In this case, overwrite is not allowed, so a new filename is automatically generated. "MYFILE.DAT, and "MYFIL1.DAT" through "MYFI24.DAT" already exist, so "MYFI25.DAT" is used. Up to 100 versions of a file are allowed before an exception is generated.

The table below illustrates a typical exchange between the POP3 Control and Mail Server.

PowerTCP API Call	POP3 Log (PowerTCP sends)	Reply from POP3 Server	Pop3 Event LastCommand Status
Action = CONNECTCOMM	'Connect'	+OK UCB Pop server (version)	POP3_CONNECT
	POP3_REPLY_POS		
	USER test	+OK Password required for test.	POP3_USER
	POP3_REPLY_POS		
	PASS xxxx	+OK test has 5 message(s) (860 octets).	POP3_PASS
	POP3_REPLY_POS		
Action = GET_POP3_STATUS	STAT	+OK 5 86000	POP3_STAT
	POP3_REPLY_POS		
Lines = 0 TopLines = 1	TOP 1 0	Return-Path: <jones@who.domain.com> Date: Sat, 6 Jan 1996 19:20:30 -0500 Received: from us.domain.com by	
who.domain.com		Message-Id: <9601070.A2@who.domain.com> To: test@who.domain.com From: doe@who.domain.com (John Doe) Subject: Test message Status: RO	
		.	POP3_TOP
	POP3_REPLY_POS		
Retrieve = 1	RETR 1	Return-Path: <jones@who.domain.com> Date: Sat, 6 Jan 1996 19:20:30 -0500 Received: from us.domain.com by	
who.domain.com		Message-Id: <9601070.A2 @who.domain.com> To: test@who.domain.com From: doe@who.domain.com (John Doe) Subject: Test message Status: RO.	
		This is a mail test	
		.	POP3_RETR
	POP3_REPLY_POS		
DeleteMsg = 1	DELE 1	+OK Message 1 has been deleted.	POP3_DELE
	POP3_REPLY_POS		
DeleteMsg = 999	DELE 999	-ERR Message 999 does not exist.	POP3_DELE
	POP3_REPLY_NEG		
Action = CLOSECOMM	QUIT	+OK POP Server signing off.	POP3_QUIT POP3_CLOSE

## POP3 Sample Application

### About POP3

The POP3 Sample Application provided in your Toolkit provides an excellent example of how to implement a POP client. This application connects to a POP server, displays the available messages, then downloads them to your workstation. The attachments are decoded on the fly.

The basic steps are: Connect, Retrieve Messages, then Close.

1) Connect to a POP Server. The following code is fired when the Retrieve Messages button is selected.

```
Sub GetM_Click ()
    POP31.AttachmentDir = dir1.Path      ' Set the Attachment Directory
    If POP31.State = CONNECTED Then
        POP31.Action = GET_POP3_STATUS
    Else
        POP31.RemoteHost = Host        ' mail server name or address
        POP31.User = User              ' user name (POP3 account)
        POP31.Password = Pass         ' user password for account
        POP31.Action = CONNECTCOMM    ' initiate connections
    End If
End Sub
```

2) Use the Pop3 Event. The Pop3 event does a great deal of work in this application. This event will receive the Pop3 Server replies that include all mail data and replies from the POP3 server.

```
Sub POP31_Pop3 (Status As Integer, LastCommand As Integer, ReplyStr As String,
FileSpec As String, Mode As String, PercentComplete As Integer)
    Dim holdres As String
    Dim holdl As String

    ' Status gives general information from the maildrop
    Select Case Status

    Case POP3_CLOSED
        ' Indication of a closed maildrop
        Text1.SelText = "Maildrop connection closed."

    Case POP3_REPLY_POS
        ' positive reply from server
        ' Output the status string for information
        Text1.SelText = ReplyStr

    Select Case LastCommand
    Case POP3_PASS
        ' This condition indicates a new connection
        ' PowerTCP sent your password
        ' Note: If you want to delete mail from the maildrop
        '   add your condition here and use the DeleteMsg
        '   Property. Then in the POP3_REPLY_POS look for
        '   LastCommand being
        '   equal to POP3_DELE to close the connection again.
        If bRetrieveMail Then
            ' Get the selected message
            POP31.Retrieve = Val(Left(List1, 5))
        Else
            ' Find the number of messages in the MailDrop
            ' Issue the STAT command
            POP31.Action = GET_POP3_STATUS
        End If
    End Select
End Sub
```

```

End If

Case POP3_STAT
' Parse the string to determine number of new messages
FirstSpace = InStr(ReplyStr, " ")
SecondSpace = InStr(FirstSpace + 1, ReplyStr, " ")
TotalMessages = Mid(ReplyStr, FirstSpace + 1, SecondSpace - (FirstSpace +
1))

nM = nMessages

' With new messages found, get the message header using
' TopLines
If TotalMessages > 0 And nM > 0 Then
    POP31.TopLines = TotalMessages ' Retrieves the top
Else
    GetM.Enabled = True ' Enable the Retrieve Button
    ' No new messages. Close maildrop.
    POP31.Action = CLOSECOMM
End If
End Select

Case POP3_REPLY_NEG ' An error from the pop server was returned
MsgBox ReplyStr
' Error with POP server. Close maildrop.
POP31.Action = CLOSECOMM

Case POP3_FILE
' Denotes an attachment status
Percent.Caption = FileSpec
Percent.Show
Percent.Text2.Width = PercentComplete * 60

Case POP3_MAIL
' Indicates data...a MAIL Message
Select Case LastCommand
Case POP3_TOP
' We have a message header
If PercentComplete = 100 Then
    ' Begin Parsing
    result = Space(5)
    LSet result = Str$(TotalMessages)
    holdres = GetPieceOfString(ReplyStr, "Return-Path:", 30)
    If InStr(holdres, "NO Return-Path:") Then
        holdres = GetPieceOfString(ReplyStr, "From", 30)
    End If
    result = result & holdres
    result = result & " "
    result = result & GetPieceOfString(ReplyStr, "Date:", 18)
    result = result & " "
    result = result & GetPieceOfString(ReplyStr, "Subject:", 30)
    List1.AddItem result
    If List1.ListIndex < nMessages Then
        ' if this is not the last message, inc row
        List1.ListIndex = List1.ListIndex + 1
    End If

    ' Continue to drive the retrieving of messages
    TotalMessages = TotalMessages - 1
    nM = nM - 1
    If TotalMessages > 0 And nM > 0 Then
        ' Not done, get next message
        POP31.TopLines = TotalMessages

```

```

        Else
            GetM.Enabled = True
            ' All done. Close maildrop
            POP31.Action = CLOSECOMM
        End If

    End If

Case POP3_RETR
    ' Indicates MAIL being retrieved
    If PercentComplete = 100 Then
        Percent.Hide
    Else
        Percent.Caption = FileSpec
        Percent.Show
        Percent.Text2.Width = PercentComplete * 60
    End If

    ' Mail Message being retrieved
    MailMsg.Text1.SelText = ReplyStr & Chr$(13) & Chr$(10)

    ' if we are done, close connection and show mail
    If PercentComplete = 100 Then
        ' We have our message. Close maildrop
        POP31.Action = CLOSECOMM
        MailMsg.Show
    End If

    End Select ' Case on LastCommand in POP3_MAIL case
End Select ' Case on the POP3 Reply Status Type

```

End Sub

**3) Retrieving a Complete Message.** Once the Headers from all of the Mail Messages are pulled down, the user can double-click on any of the messages and the complete message will be pulled down.

```

Sub List1_DblClick ()
    ' specify location for all attachment received
    POP31.AttachmentDir = dir1.Path
    ' The Message number to retrieve...
    POP31.Retrieve = Val(Left(List1, 5))
End Sub

```

## POP3 Constants

### About POP3

The following constants should be used to control and monitor POP3 activity.

#### ' Status in the POP3 event

Global Const POP3\_CLOSED = 0 ' The connection to the POP3 server is closed.

Global Const POP3\_REPLY\_POS = 1

' The POP3 server has responded with a positive "+OK", indicating a positive  
' reply to the previous message sent to the server. If a multi-line reply  
' and the end is not present in the Reply buffer, then PercentComplete will  
' be 0 to indicate this fact. If a single-line reply or includes the end of  
' a multi-line reply, then PercentComplete will be 100.

Global Const POP3\_REPLY\_NEG = 2

' The POP3 server has responded with a negative "-ERR", indicating a  
' negative reply to the previous message sent to the server.

Global Const POP3\_MAIL = 3

' The Reply buffer holds part or all of a mail message header and/or body.

Global Const POP3\_FILE = 4

' The Reply buffer holds part or all of a file. The Mode parameter may be  
' checked for the encoding method used for the file. The FileSpec parameter  
' specifies the file being filled if the AttachmentDir property was  
' specified.

Global Const POP3\_HEADER = 5

' The Reply buffer holds a complete mail header. Occurs only in response to Retrieve requests.

#### ' LastCommand in the Pop3 event

Global Const POP3\_APOP = 0 ' APOP Authentication sent

Global Const POP3\_DELE = 1 ' Delete Message

Global Const POP3\_LIST = 2 ' List a particular MAIL message(s)

Global Const POP3\_NOOP = 3 ' No Action

Global Const POP3\_PASS = 4 ' Password

Global Const POP3\_QUIT = 5 ' Quit the MAIL

Global Const POP3\_RSET = 6 ' Reset the POP Server

Global Const POP3\_RETR = 7 ' Retrieve a MAIL Message

Global Const POP3\_STAT = 8 ' Check Status

Global Const POP3\_TOP = 9 ' TOP of Mail Message

Global Const POP3\_UIDL = 10 ' Not Implemented

Global Const POP3\_USER = 11 ' Username

Global Const POP3\_CONNECT = 12 ' Reply received after connecting to server.

#### ' Actions for POP3

Global Const CONNECTCOMM = 0 ' Connect to server

Global Const CLOSECOMM = 2 ' Close connection

Global Const ABORTCOMM = 3 ' Abort connection

Global Const GET\_POP3\_STATUS = 5 ' Get the POP3 Status

Global Const SEND\_NOOP\_POP3 = 11 ' Send a NOOP to the POP server

Global Const SEND\_RESET\_POP3 = 12 ' Reset the POP3 Server

## POP3 Custom Control

[About POP3](#)



P16POP6.VBX

### Object Type

PowerTCP\_POP3

The properties and events for this control are listed in the following table. Properties and events that apply *only* to this control, or which require special consideration when used with it, are marked with an asterisk (\*). They are documented in the Properties and Events section.

### Properties

<u>*Action</u>	<u>*Lines</u>	<u>*OemLicense</u>	<u>*State</u>
<u>*AttachmentDir</u>	<u>*List</u>	Parent	Tag
<u>*DeleteMsg</u>	<u>*LocalDotAddr</u>	<u>*Password</u>	Top
<u>*Flags</u>	<u>*LocalName</u>	<u>*RemoteHost</u>	<u>*TopLines</u>
hWnd	<u>*LocalPort</u>	<u>*RemotePort</u>	<u>*Uidl</u>
Index	<u>*MailFileSpec</u>	<u>*Retrieve</u>	<u>*User</u>
Left	Name	<u>*Secret</u>	

### Events

<u>*Connect</u>	<u>*File</u>	<u>*Pop3</u>
<u>*Exception</u>	<u>*Log</u>	<u>*Header</u>

## About UDP

UDP means User Datagram Protocol. UDP is built upon IP just as TCP is. However, there are differences in the ways that UDP and TCP use IP. UDP provides an *unreliable* transmission of packets between two computers. This differs from TCP in two ways: TCP is reliable, and it provides a data stream instead of individual packets.

UDP is unreliable because when data is sent, the sender has no way of knowing if the data reached its destination or not. UDP packets are sent and received individually, and not split apart or recombined. A data stream (used by TCP) would send the data in arbitrary packets, ensuring only that data would be received in the same order in which it was sent.

In UDP, no true connections are made. A computer simply opens a UDP port and waits for data. Computers can send data to each other on a particular port. This is in contrast to TCP, where a connection must be established before data is transferred.

UDP is the base protocol for upper-layer protocols such as SNMP (Simple Network Management Protocol) and TFTP (Trivial File Transfer Protocol). These protocols are implemented using UDP.

The only features UDP provides over IP are port numbers and checksums (which are optional, and not always implemented). UDP data is sent and received on specified ports, which allow a computer to handle multiple connections at the same time on different ports.

For more information on UDP, refer to RFC 768.

[UDP and PowerTCP](#)

[UDP Custom Control](#)





## **UDP and PowerTCP**

### About UDP

All PowerTCP custom controls with the exception of UDP are compatible with all versions of Microsoft Visual Basic (including version 1.0). However, the UDP control is compatible only with versions 2.0, 3.0 and 4.0. This means that the UDP custom control cannot be used with Visual Basic 1.0, Microsoft Visual C++ (using MFC), PowerBuilder, and Delphi (there is a separate Delphi PowerTCP Toolkit, however, with Delphi components instead of VBXs). However, all other PowerTCP custom controls will work in these environments.

# UDP Custom Control

[About UDP](#)



P16UDPB5.VBX

## Object Type

PowerTCP\_UDP

The properties and events for this control are listed in the following table. Properties and events that apply *only* to this control, or which require special consideration when used with it, are marked with an asterisk (\*). They are documented in the following sections.

### Properties

<u>*Action</u>	<u>*LocalDotAddr</u>	<u>*OemLicense</u>	Tag
<u>*DataTag</u>	<u>*LocalPort</u>	<u>*RemoteHost</u>	Top
<u>*Flags</u>	Left	<u>*RemotePort</u>	
hWnd	Name	<u>*Send</u>	
Index	Parent	<u>*State</u>	

### Events

<u>*Connect</u>	<u>*Exception</u>	<u>*Recv</u>	<u>*Send</u>
-----------------	-------------------	--------------	--------------

## About SNMP

SNMP stands for Simple Network Management Protocol. This section describes SNMP (version 1) only briefly because of its complexity. See RFCs 1213, 1155, and 1157 for more information, or one of the books listed in [Additional Resources](#).

SNMP provides for basic network monitoring and management functions between a management station and managed nodes through applications located on the nodes, called agents. These agents maintain a database of information about managed nodes and when queried by the manager, retrieve data from the database and send it back to the manager in a response. SNMP over TCP/IP uses UDP packets as its transport for these queries and responses, as well as for traps. Traps are sent to the management station when certain events occur, like when a node comes up or is shut down.

PowerTCP supports the encoding and decoding of these packets, greatly simplifying the task of getting and setting SNMP object values. Use PowerTCP to create **BOTH** SNMP Management Applications **AND** SNMP Agent Applications. PowerTCP supports SNMP version 1.

[SNMP Messages](#)

[SNMP and PowerTCP](#)

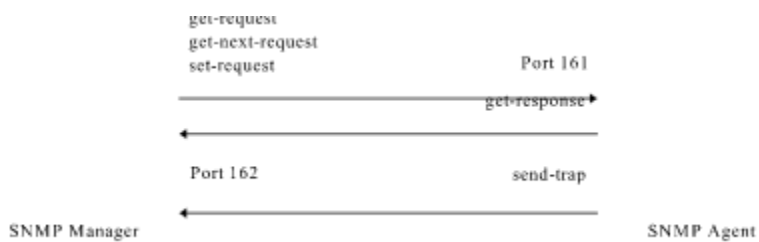
[SNMP Custom Control](#)

## SNMP Messages

### About SNMP

Five different types of messages are exchanged between the manager and its managed nodes.

1. get-request - retrieves the value of one or more objects from an agent's MIB.
2. get-next-request - retrieves the next value of one or more objects from an agent's MIB.
3. set-request - sets the value of one or more objects in an agent's MIB.
4. get-response - returns the value of one or more objects in response to a manager's request.
5. trap - notifies a manager when an event occurs on an agent.



## **SNMP and PowerTCP**

### About SNMP

PowerTCP makes building a network management application in Microsoft Windows a much easier task by providing an asynchronous, event driven interface to the messages used by SNMP. The PowerTCP SNMP interface:

1. Handles the building and encoding of outbound messages through the ActionAction\_SNMP property.
2. Handles the parsing and decoding of inbound messages and notifies the application by generating a RecvSnmprRecvSnmpr Event.
3. Handles the parsing and decoding of inbound trap messages and notifies the application by generating a RecvTrapRecvTrap event.
4. Greatly simplifies set-up and tear-down operations.

# SNMP Custom Control

[About SNMP](#)



P16SNMB5.VBX

## Object Type

PowerTCP\_SNMP

The properties and events for this control are listed in the following table. Properties and events that apply *only* to this control, or which require special consideration when used with it, are marked with an asterisk (\*). They are documented in the following sections.

### Properties

<u>*Action</u>	hWnd	<u>*NodeDotAddr</u>	<u>*RemotePort</u>
<u>*Community</u>	Index	Parent	<u>*RequestID</u>
<u>*Enterprise</u>	<u>*LocalDotAddr</u>	<u>*ObjectID</u>	<u>*SpecialTrap</u>
<u>*ErrorIndex</u>	<u>*LocalPort</u>	<u>*ObjectType</u>	<u>*State</u>
<u>*ErrorStatus</u>	Left	<u>*ObjectValue</u>	<u>*TimeStamp</u>
<u>*Flags</u>	Name	<u>*OemLicense</u>	Tag
<u>*GeneralTrap</u>	<u>*nObjects</u>	<u>*RemoteHost</u>	Top

### Events

<u>*Connect</u>	<u>*RecvSnmp</u>	<u>*Send</u>
<u>*Exception</u>	<u>*RecvTrap</u>	

## **About TFTP**

TFTP stands for Trivial File Transfer Protocol. It is a protocol used for transferring files using the UDP protocol. This is in contrast to FTP, which uses TCP. TFTP was originally designed to be a small, simple method for file transfer during bootstrapping.

TFTP provides only rules for uploading and downloading files. Because it is based on UDP, it is not necessarily reliable, and so data corruption can occur if UDP checksums are not enabled by the operating system.

TFTP does not provide any kind of security. Therefore, TFTP servers generally restrict access to files in a TFTP directory. TFTP does not include support for transferring directory listings, or many other features which FTP implements. TFTP is also generally slower than FTP file transfers because every packet is acknowledged. The standard TFTP port is 69.

TFTP is more useful than FTP when it is important to minimize usage of system resources, and when simple file transfers are needed without dealing with security features like username and password. It may also be quicker than FTP for small file transfers, as less setup is required for each transfer.

For more information on TFTP, see RFC 1350.

[TFTP and PowerTCP](#)

[TFTP Custom Control](#)

## **TFTP and PowerTCP**

[About TFTP](#)

PowerTCP supports TFTP revision 2.



## TFTP Custom Control

[About TFTP](#)



P16TFTB5.VBX

### Object Type

PowerTCP\_TFTP

The properties and events for this control are listed in the following table. Properties and events that apply *only* to this control, or which require special consideration when used with it, are marked with an asterisk (\*). They are documented in the following sections.

### Properties

<u>*AbortTransfer</u>	<u>*LocalDotAddr</u>	Parent	<u>*State</u>
<u>*Action</u>	<u>*LocalFileSpec</u>	<u>*OemLicense</u>	<u>*TimeoutInterval</u>
<u>*Flags</u>	Left	<u>*RemoteFileSpe</u>	Tag
hWnd	<u>*MaxRetries</u>	<u>C</u>	Top
Index	Name	<u>*RemoteHost</u>	
		<u>*ServerPort</u>	

### Events

<u>*Connect</u>	<u>*Exception</u>	<u>*Tftp</u>
-----------------	-------------------	--------------

## Abort Property

### Applies To

[FTP](#).

### Description

Sends an *abort* (ABOR) command. Tells the server to abort the previous FTP file transfer command (APPE, STOR, RETR or STOU) and any associated transfer of data. The control connection is not closed by the server, and the data connection is closed. This property is not displayed at design time, and is write-only at run time.

### Usage

[*form.*][*control.*]**Abort** = **True**

### Remarks

Successful completion is indicated by checking the Status parameter in the Reply Event.

### Data Type

**Integer** (Boolean)

### See Also

[Reply](#) Event.

## **AbortTransfer Property**

### **Applies To**

TFTP.

### **Description**

Aborts the file transfer specified. The TransferID is reported in the TFTP event.

### **Usage**

[*form.*][*control.*]**AbortTransfer** = *TransferID*

### **Data Type**

**Long**

## **Account Property**

### **Applies To**

FTP.

### **Description**

The account used for logging in to an FTP host.

### **Usage**

[*form.*][*control.*]**Account** = *stringexpression*

### **Remarks**

Only some FTP hosts require an account identifier. Most require only a username and password. This should be set before the LoginHost property is set.

### **Data Type**

**String**

### **See Also**

Password Property; User Property.

## Action Property (POP3)

### Applies To

POP3.

### Description

The Action property sets the state of the control, ends a connection with another computer, or sends POP3 messages without parameters. Properties are used to send POP3 messages with parameters. This property is not displayed at design time, and write-only at run time.

### Usage

[*form.*][*control.*]**Action** = *setting*

### Remarks

The Action property settings are:

Constant	Description
CONNECTCOMM = 0	Sets the state to Connecting. This connects you with a remote computer. You must have previously specified the RemoteHost, User and Password properties. You will receive a Connect event when the channel is created.
CLOSECOMM = 2	Closes the connection with another computer by sending a QUIT message. You are notified with a Pop3 Event (Status = POP3_CLOSED) when a connection is closed.
ABORTCOMM = 3	Terminates TCP connection with no delay. Preferably, you should use the CLOSECOMM setting instead. You are notified with a Pop3 Event (Status = POP3_CLOSED) when a connection is closed.
GET_POP3_STATUS = 5	The POP3 server issues a positive response with a line containing information for the maildrop. This line is called a "drop listing" for that maildrop. In order to simplify parsing, all POP3 servers are required to use a certain format for drop listings. The positive response consist of "+OK" followed by a single space, the number of messages in the maildrop, a single space, and the size of the maildrop in octets. The protocol makes no requirement on what follows the maildrop size. Minimal implementations end that line of the response with a CRLF pair. More advanced implementations may include other information. Note that messages marked as deleted are not counted in either total.
SEND_NOOP_POP3 = 11	Sends a NOOP message to the POP3 server.
SEND_RESET_POP3 = 12	Sends a RESET message to the POP3 server. If any messages have been marked as deleted by the POP3 server, they are unmarked. The POP3 server then replies with a positive response.

### Data Type

#### Integer

#### See Also

Connect Event; Exception Event; Pop3 Event; State Property.

## Action Property (SMTP)

### Applies To

SMTP.

### Description

The Action property makes a connection, disconnects, or sends mail. This property is not displayed at design time, and write-only at run time.

### Usage

[*form.*][*control.*]**Action** = *setting*

### Remarks

The Action property settings are:

Constant	Description
----------	-------------

CONNECTCOMM = 0	Sets the state to Connecting. This connects you with a remote computer. You must have previously specified the RemoteHost property. If the connect is successful, you will receive a Connect event.
CLOSECOMM = 2	Closes the connection with another computer by sending a QUIT message. When an active connection is closed, you are notified with an Smtplib Event where the Status is SMTP_CLOSED.
ABORTCOMM = 3	Terminates TCP connection with no delay. Preferably, you should use the CLOSECOMM setting instead. When an active connection is closed, you are notified with an Smtplib Event where the Status is SMTP_CLOSED.
RESET_MAIL = 4	Sends a Reset message to the mail host, causing it to terminate the current mail transaction.
SEND_MAIL = 5	Initiates a mail message. You should have previously set the following properties: Sender, Recipient, HeaderDate, HeaderSubject, HeaderTo, and Message.
SEND_SEND = 6	Initiates a send message. You should have previously set the following properties: Sender, Recipient, HeaderDate, HeaderSubject, HeaderTo, and Message.
SEND_SOML = 7	Initiates a send or mail message. You should have previously set the following properties: Sender, Recipient, HeaderDate, HeaderSubject, HeaderTo, and Message.
SEND_SAML = 8	Initiates a send and mail message. You should have previously set the following properties: Sender, Recipient, HeaderDate, HeaderSubject, HeaderTo, and Message.
SEND_NOOP_MAIL = 11	This command does not affect any parameters or previously entered commands. It specifies no action other than that the receiver send an OK reply. This command has no effect on the reverse-path buffer, the forward-path buffer, or the mail data buffer.

### Data Type

#### Integer

### See Also

Connect Event; Exception Event; Smtplib Event; State Property.

## Action Property (SNMP)

### Applies To

SNMP.

### Description

The Action property initiates a change-of-state and is also used to generate get-request, get-next-request and set-request SNMP packets. This property is not displayed at design time, and write-only at run time.

### Usage

[*form.*][*control.*]**Action** = *setting*

### Remarks

The Action property settings are:

Constant	Description
CONNECTCOMM = 0	Allocates socket resources and establishes a local SNMP session. If successful, you will receive a <u>Connect</u> event. If not, you will receive an <u>Exception</u> event.
CLOSECOMM = 2	Closes the session after any outstanding packets are sent.
ABORTCOMM = 3	Aborts the session.
SNMP_GET_REQUEST = 4	Generates and sends a get-request SNMP packet.
SNMP_GET_NEXT_REQUEST = 5	Generates and sends a get-next-request SNMP packet.
SNMP_GET_RESPONSE = 6	Generates and sends a get_response SNMP packet.
SNMP_SET_REQUEST = 7	Generates and sends a set-request SNMP packet.
SNMP_TRAP_MESSAGE = 8	Generates and sends a trap SNMP packet.

### Data Type

**Integer**

### See Also

State Property; Connect Event; Exception Event; RecvSnmp Event

## Action Property (TCP, Telnet)

### Applies To

TCP, Telnet.

### Description

The Action property makes a connection, disconnects, or listens for connections. This property is not displayed at design time, and write-only at run time.

### Usage

[*form.*][*control.*]**Action** = *setting*

### Remarks

The Action property settings are:

Setting	Description
---------	-------------

CONNECTCOMM = 0	Connect - Attempts to make a connection with a remote computer. You must have previously specified the RemoteHostRemoteHost and RemotePortRemotePort properties. If the connect is successful, you will receive a <u>Connect</u> event.
-----------------	---

LISTENCOMM = 1	Listen - Begins to listen for remote computers trying to connect. When a computer sends a message to the control that it wants to connect, you must assign the connection to a new control. For more information on the Listening state, see the <u>Listen</u> and <u>Accept</u> events. If entering Listening state was successful, you will receive a <u>Listen</u> event.
----------------	--

CLOSECOMM = 2	Close - Closes the connection (after flushing all sent data) or stops listening for new connections. When an active connection is closed, you are notified with a <u>Recv</u> Event where the length of the RecvData parameter is 0.
---------------	--

ABORTCOMM = 3	Abort - Terminates the TCP connection with no delay. Preferably, you should use CLOSECOMM instead. When an active connection is closed, you are notified with a <u>Recv</u> Event where the length of the RecvData parameter is 0.
---------------	--

### Example

The following lines of code show how to make a connection to the computer *eagle1.dart.com* on the Telnet port.

```
Telnet1.RemoteHost = "eagle1.dart.com"  
Telnet1.RemotePort = 23      ' The standard Telnet port  
Telnet1.Action = CONNECTCOMM ' We will receive the  
                             ' Connect event when  
                             ' it is established
```

### Data Type

#### Integer

### See Also

Accept Event; State Property; Connect Event; Exception Event; Listen Event.

## Action Property (TFTP)

### Applies To

TFTP.

### Description

The Action property initiates a change-of-state. This property is not displayed at design time, and write-only at run time.

### Usage

[*form.*][*control.*]**Action** = *setting*

### Remarks

The Action property settings are:

<b>Setting</b>	<b>Description</b>
----------------	--------------------

CONNECTCOMM = 0	Allocates socket resources and establishes a TFTP session. If successful, you will receive a <u>Connect</u> event. If not, you will receive an <u>Exception</u> event.
-----------------	--

LISTENCOMM = 1	Sets the state to Connected. This establishes a local TFTP server. If successful, you will receive a <u>Connect</u> event. If not, you will receive an <u>Exception</u> event.
----------------	--

CLOSECOMM = 2	Closes the session after all file activity is complete. You will receive the <u>Tftp</u> event with <i>Op</i> = TFTP_CLOSED as confirmation.
---------------	--

ABORTCOMM = 3	Immediately aborts all file transfer activity and closes. You will receive the <u>Tftp</u> event with <i>Op</i> = TFTP_CLOSED as confirmation.
---------------	--

GET_NETASCII = 8	Initiate a get file operation using NETASCII mode.
------------------	--

GET_OCTET = 9	Initiate a get file operation using OCTET mode.
---------------	---

PUT_NETASCII = 10	Initiate a put file operation using NETASCII mode.
-------------------	--

PUT_OCTET = 11	Initiate a put file operation using OCTET mode.
----------------	---

### Data Type

### Integer

### See Also

State Property; Connect Event; Exception Event.



## Action Property (UDP)

### Applies To

UDP.

### Description

The Action property initiates a change of state. This property is not displayed at design time, and write-only at run time.

### Usage

[*form.*][*control.*]**Action** = *setting*

### Remarks

The Action property settings are:

<b>Constant</b>	<b>Description</b>
-----------------	--------------------

CONNECTCOMM = 0	Makes a connection and establishes a local UDP session. If successful, you will receive a ConnectConnect_UDP_SNMP event. If not, you will receive an ExceptionException event.
-----------------	--

CLOSECOMM = 2	Closes the session after outstanding buffers are sent. The RecvRecv_Event_UDP event is called with RecvData = "" as confirmation.
---------------	---

ABORTCOMM = 3	Closes the session immediately, destroying any outstanding buffers. The RecvRecv_Event_UDP event is called with RecvData = "" as confirmation.
---------------	--

### Data Type

#### Integer

### See Also

Accept Event; State Property; Connect Event; Exception Event; Listen Event.

## Allocate Property

### Applies To

FTP.

### Description

Sends an *allocate* (ALLO) command. May be required by some servers to reserve sufficient storage to accommodate the new file to be transferred. This property is not displayed at design time, and is write-only at run time.

### Usage

[*form.*][*control.*]**Allocate** = *stringexpression*

### Remarks

Set the Allocate Property to the maximum file size and maximum record size values desired. Each value should be separated by a space, with the second value optional. **Do not** put an 'R' between the two values. For example:

```
FTP1.Allocate = "1024 512"
```

Successful completion is indicated by checking the Status parameter in Reply Event.

### Data Type

### String

### See Also

Reply Event.

## **AnswerBack Property**

### **Applies To**

VT.

### **Description**

Sets or returns a string expression that is sent by the control whenever an ENQ is received.

### **Usage**

[*form.*][*control.*]**AnswerBack** = *stringexpression*

### **Data Type**

**String**

## **Appe Property**

### **Applies To**

FTP.

### **Description**

Sends an *append* with create (APPE) command. Similar to the Store Property, except that if the file exists then additional data is appended to it. This property is not displayed at design time, and is write-only at run time.

### **Usage**

[*form.*][*control.*]**Appe** = *stringexpression (PathName)*

### **Remarks**

Setting the Appe Property to a pathname will set-up a data connection and deposit all data sent across it into the pathname specified.

Successful completion is indicated by checking the Status parameter in Reply Event.

### **Data Type**

**String**

### **See Also**

Send Event; Send Property; Reply Event.

## **AttachmentDir Property**

### **Applies To**

POP3.

### **Description**

Specifies the path to be used for attached files. Should be set before the Retrieve property is used.

### **Usage**

[*form.*][*control.*]**AttachmentDir**[ = *stringexpression*]

### **Remarks**

Since PowerTCP supports MIME and UUENCODE specifications, attached files must be spooled into a directory. This property sets the location to be used.

### **Data Type**

**String**

## **Attachments Property**

### **Applies To**

SMTP.

### **Description**

Sets or returns a string of comma-delimited file specifications.

### **Usage**

[*form.*][*control.*]**Attachments**[ = *stringexpression*]

### **Remarks**

This property must be set before invoking Action = SEND\_MAIL if attachments are to be included. There is a one-to-one correspondence between each attachment file specified with the Attachments property and the encoding modes specified by the AttachTypes property.

### **Data Type**

**String**

### **See Also**

AttachTypes Property.

## **AttachTypes Property**

### **Applies To**

SMTP.

### **Description**

Sets or returns a string of comma-delimited attachment encoding modes.

### **Usage**

[*form.*][*control.*]**AttachTypes**[ = *stringexpression*]

### **Remarks**

A NULL value may be used if no encoding is required, otherwise the string must contain an encoding mode for each file attachment in the Attachments property.

The table below lists the valid encoding modes.

<b>Setting</b>	<b>Description</b>
----------------	--------------------

---

M	MIME (Base64 Encoding)
U	UU Encoding
N	No Encoding

### **Data Type**

#### **String**

### **See Also**

Attachments Property.

## **AutoOption Property**

### **Applies To**

Telnet.

### **Description**

Sets or returns whether automatic option negotiation is enabled.

### **Usage**

[*form.*][*control.*]**AutoOption**[ = {**True**|**False**} ]

### **Remarks**

When True, there is no need to respond to the Cmd event, as option negotiation is taken care of. The automatic option negotiation responds with *Will Suppress Go Aheads*, *Do Suppress Go Aheads*, and *Do Echo* commands. If you wish to support additional features, you should set this property to FALSE and write custom code in the Cmd event.

### **Data Type**

**Integer** (Boolean)

### **See Also**

Cmd Event.



## **AutoPrint Property**

### **Applies To**

VT.

### **Description**

Sets or returns whether auto print mode is on.

### **Usage**

[*form.*][*control.*]**AutoPrint**[ = {**True**|**False**} ]

### **Remarks**

When True, the control will print displayed lines to the default printer.

### **Data Type**

**Integer** (Boolean)

## **AutoRepeat Property**

### **Applies To**

VT.

### **Description**

Sets or returns whether depressed keys will generate more than one character.

### **Usage**

[*form.*][*control.*]**AutoRepeat**[ = {**True**|**False**} ]

### **Remarks**

If False, only a single character will be generated when a key is depressed for a time interval.

If True, multiple characters will be generated until the key is released.

### **Data Type**

**Integer** (Boolean)

## **AutoWrap Property**

### **Applies To**

VT.

### **Description**

Sets or returns whether display text prints at the last column or automatically wraps to the next line when more data is received than fits in one row.

### **Usage**

[*form.*][*control.*]**AutoWrap**[ = {**True|False**} ]

### **Remarks**

If False, characters received after the right margin are overwritten into the last character position of the current line.

If True, a character received after the right margin automatically appears in the first character position of the next line.

### **Data Type**

**Integer** (Boolean)

## **BackColor Property**

### **Applies To**

VT.

### **Description**

Sets or returns the color used for screen background.

### **Usage**

[*form.*][*control.*]**BackColor** [ = *numericexpression* ]

### **Remarks**

RGB and system colors are supported. The new color is immediately used.

### **Examples**

The following are all valid statements:

```
VT1.BackColor = RGB(255, 0, 0)      ' Red
VT1.BackColor = QBColor(12)        ' Red
VT1.BackColor = &H8000000F         ' System button color
                                   ' (normally gray)
```

### **Data Type**

**Long**

## **Bell Property**

### **Applies To**

VT.

### **Description**

Sets or returns whether the audio bell is enabled or disabled.

### **Usage**

[*form.*][*control.*]**Bell**[ = {**True**|**False**} ]

### **Remarks**

If True, the ASCII BEL character will cause the system bell to sound. In addition, some actions (like attempting to scroll past the scroll range) will cause the bell to sound.

If False, the system bell is disabled.

### **Data Type**

**Integer** (Boolean)

## **BoldColor Property**

### **Applies To**

VT.

### **Description**

Sets or returns the color used for characters displayed with the bold attribute.

### **Usage**

[*form.*][*control.*]**BoldColor** [ = *numericexpression* ]

### **Remarks**

RGB and system colors are supported. The new color is immediately used.

### **Examples**

The following are all valid statements:

```
VT1.BoldColor = RGB(255, 0, 0)      ' Red
VT1.BoldColor = QBColor(12)        ' Red
VT1.BoldColor = &H8000000F         ' System button color
                                   ' (normally gray)
```

### **Data Type**

**Long**

## **BufferRows Property**

### **Applies To**

VT.

### **Description**

Sets or returns the number of lines used for scroll-back buffering.

### **Usage**

[*form.*][*control.*]**BufferRows**[=*numericexpression* ]

### **Remarks**

The BufferRows can be reset at any time; data will be preserved if possible.

### **Data Type**

**Integer**

### **See Also**

Cols Property; Rows Property; Text Property.

## **BufferSize Property**

### **Applies To**

FTP.

### **Description**

Sets the size of the individual blocks of data which are sent over the data connection during a file transfer.

### **Usage**

[*form.*][*control.*]**BufferSize**[= *numericexpression*]

### **Remarks**

The BufferSize property can be used to optimize the transfer rate between computers. A value which is too low can slow down a transfer, but so can a value which is too high. In addition, the optimal value may vary between machines. A value between six thousand and twelve thousand usually works well.

The BufferSize property only applies to file transfers which are handled by PowerTCP (using the LocalFileSpec property). If your program is manually "feeding" the data to the FTP control, then the size of the strings the control is given is essentially the buffer size.

### **Data Type**

**Integer**



## **ChDir Property**

### **Applies To**

FTP.

### **Description**

Sends the *Change Working Directory* (CWD) command. Use this property to change the current working directory. This property is not displayed at design time, and is write-only at run time.

### **Usage**

[*form.*][*control.*]**ChDir** = *stringexpression* (*PathName*)

### **Remarks**

Successful completion is indicated by checking the Status parameter in Reply Event.

### **Data Type**

**String**

### **See Also**

Reply Event.

## **ChDirUp Property**

### **Applies To**

FTP.

### **Description**

Sends a *Change to Parent Directory* (CDUP) command. Use this property to change the current working directory to the parent directory. This property is not displayed at design time, and is write-only at run time.

### **Usage**

[*form.*][*control.*]**ChDirUp** = **True**

### **Remarks**

Successful completion is indicated by checking the Status parameter in Reply Event.

### **Data Type**

**Integer** (Boolean)

### **See Also**

Reply Event.

## **CloseControl Property**

### **Applies To**

FTP.

### **Description**

This property can be set to close the FTP control connection. Under normal circumstances, Logout should be used instead. This property is not displayed at design time, and is write-only at run time.

### **Usage**

`[form.][control.]CloseControl = True`

### **Data Type**

**Integer** (Boolean)

### **See Also**

Logout Property.

## **CloseData Property**

### **Applies To**

FTP.

### **Description**

This property is used to close the FTP data connection. This property is not displayed at design time, and is write-only at run time.

### **Usage**

*[form.]**[control.]***CloseData = True**

### **Remarks**

For Store, StoreUnique, and Append commands you can use the Send property to feed the file to the server. When complete, CloseData should be set to True to signal the end of the file transfer.

### **Data Type**

**Integer** (Boolean)

### **See Also**

Flags Property.

## **Cmd Property**

### **Applies To**

Telnet.

### **Description**

The Cmd property sends a Telnet command to the other end of the connection. This property is not displayed at design time, and write-only at run time.

### **Usage**

*[form.][control.]Cmd = numericexpression*

### **Remarks**

The Cmd property is used only when AutoOption is set to False, since it is used during option negotiation.

For more information on option negotiation, see the Option Negotiation section of the Telnet chapter.

### **Data Type**

**Integer**

### **See Also**

Cmd Event.

## **Cols Property**

### **Applies To**

VT.

### **Description**

Sets or returns the number of columns being used for the display.

### **Usage**

[*form.*][*control.*]**Cols**[= *numericexpression*]

### **Remarks**

Cols can be reset at any time; data will be preserved if possible. Only 80 and 132 are valid values. Attempting to set Cols to other values will result in a runtime error.

### **Data Type**

**Integer**

### **See Also**

BufferRows Property; Rows Property; Text Property.

## **ColWidth Property**

### **Applies To**

VT.

### **Description**

Returns the width of a column according to the measurement units set up by the container object (normally the form).

### **Usage**

[*form.*][*control.*]**ColWidth**

### **Remarks**

Read-only at runtime. When multiplied by the number of columns (80 or 132), the application can compute the width of the control necessary to view one screen of data.

### **Example**

The following line of code makes the control the optimum width with no truncation or extra space (the scroll bar is normally 255 twips wide):

```
VT1.Width = VT1.Cols * VT1.ColWidth + 255
```

### **Data Type**

**Integer**

### **See Also**

Cols Property.

## Command Property

### Applies To

FTP.

### Description

Sends an FTP command string for transmission over the control connection. This property is not displayed at design time, and is write-only at run time.

### Usage

[*form.*][*control.*]**Command** = *stringexpression*

### Remarks

This property is used if you want to bypass the library functions that handle command string creation and reply synchronization and checking. It may be useful if the FTP host is not responding as expected by the protocol, and you wish to troubleshoot the operation. PowerTCP will terminate the command with a CR/LF pair.

When used, the PowerTCP control has no way of knowing what the current synchronization is, and uses FTP\_UNKNOWN as the Status value for the Reply Event.

### Example

A sample command to set the host and port would be:

```
FTP1.Command = "PORT 129,229,1,5,0,200"
```

### Data Type

### String

### See Also

Reply Event.



## Community Property

### Applies To

SNMP.

### Description

Specifies the community name to use for the get-request, get-next-request, or set-request messages. The value is set each time a RecvSnmp or RecvTrap Event is called. Typical values are "public" or "private".

### Usage

[*form.*][*control.*]**Community**[ = *stringexpression*]

### Remarks

Must match the community name that is specified on the agent.

### Data Type

**String**

## **Cursor Property**

### **Applies To**

VT.

### **Description**

Sets or returns whether the cursor is displayed or not. The location of the cursor is unaffected.

### **Usage**

[*form.*][*control.*]**Cursor**[ = {**True**|**False**} ]

### **Remarks**

If True, displays the cursor. If 0, hides the cursor. Note that the cursor is only shown when the control has the focus.

### **Data Type**

**Integer** (Boolean)

## **CursorCol Property**

### **Applies To**

VT.

### **Description**

Returns the current cursor column position (between 1 and either 80 or 132). Read-only at runtime.

### **Usage**

[*form.*][*control.*]**CursorCol**

### **Data Type**

### **Integer**

### **See Also**

Cols Property; CursorRow Property.

## CursorKeys Property

### Applies To

VT.

### Description

Sets or returns whether the cursor keys send ANSI cursor control sequences or application control functions.

### Usage

[*form.*][*control.*]**CursorKeys**[= setting]

<b>Setting</b>	<b>Description</b>
0	Normal Cursor Keys. Cursor keys send ANSI cursor control sequences (up, down, left and right).
1	Application Cursor Keys. Cursor keys send application program control functions.

### Data Type

**Integer** (Enumerated)

## **CursorRow Property**

### **Applies To**

VT.

### **Description**

Returns the current cursor row position.

### **Usage**

[*form.*][*control.*]**CursorRow**

### **Remarks**

Since the Rows property can be changed, the range of CursorRow is between 1 and the **Rows Property**.

### **Data Type**

**Integer**

### **See Also**

Rows Property; CursorCol Property.

## CursorStyle Property

### Applies To

VT.

### Description

Sets or returns the cursor style displayed.

### Usage

[*form.*][*control.*]**CursorStyle**[= setting%]

Setting	Description
---------	-------------

---

0	Block cursor.
---	---------------

1	Underline cursor.
---	-------------------

### Data Type

**Integer** (Enumerated)

## **DataTag Property**

### **Applies To**

TCP, Telnet, SMTP, UDP.

### **Description**

A multi-purpose value which is passed back to the Send event after setting the Send, SendByte, or SendString properties. This property is not displayed at design time.

### **Usage**

[*form.*][*control.*]**DataTag**[ = *numericexpression*]

### **Remarks**

The DataTag property is comparable to the Tag property in many standard controls - the developer may use it for many different purposes. The DataTag property is slightly different, however. When you set the Send property to send a string, and then receive the Send event, the value of DataTag at the time the string was sent is passed back to you as an argument. This makes it useful as a tag to mark sent data.

### **Data Type**

**Long**

### **See Also**

Send Event; Send Property.

## **Dele Property**

### **Applies To**

FTP.

### **Description**

Sends a *Delete* (DELE) command. Causes the specified file to be deleted at the server site. This property is not displayed at design time, and is write-only at run time.

### **Usage**

[*form.*][*control.*]**Dele** = *stringexpression (PathName)*

### **Remarks**

Set the Dele Property to the pathname of the file you wish to delete. For example:

```
FTP1.Dele = "filename.hog" ' free up some disk space
```

### **Data Type**

**String**

### **See Also**

Reply Event.



## DeleteMsg Property

### Applies To

POP3.

### Description

Set the Delete Property to a message number to send a POP3 DELE message. This property is not visible at design-time and is write-only at run-time.

### Usage

[*form.*][*control.*]**DeleteMsg** = [ *numericexpression* ]

### Remarks

The POP3 server marks the message as deleted. Any future reference to the message-number associated with the message in a POP3 command generates an error. The POP3 server does not actually delete the message until **Action=CLOSECOMM** is set. The reply from the POP3 server causes the **Pop3 event** to fire.

### Data Type

**Integer**

## Display Property

### Applies To

VT.

### Description

When set to a string of data, interprets and displays that data according to the current Terminal property setting.

### Usage

[*form.*][*control.*]**DisplayString**[ = *stringexpression*]

### Remarks

Write only at runtime. If the Enabled Property is False, setting this property has no effect.

### Important

This is a Visual Basic 2.0 property, and can contain NULL characters that will be ignored by the emulator. For Delphi, PowerBuilder, and Microsoft Visual C++, you should use the DisplayString property.

### Example

The following example displays all data received from a Telnet control:

```
Sub Telnet1_Recv (RecvData As String)
    VT1.Display = RecvData
End Sub
```

### Data Type

#### String

### See Also

DisplayString Property; Enabled Property; KeyPress Event.

## DisplayString Property

### Applies To

VT.

### Description

When set to a string of data, interprets and displays that data according to the current Terminal property setting.

### Usage

[*form.*][*control.*]**Display**[ = *stringexpression*]

### Remarks

Write only at runtime. If the Enabled Property is False, setting this property has no effect.

### Important

This property should be used instead of Display only in PowerBuilder, Delphi, and Visual C++, as they use null-terminated strings. This is a Visual Basic 1.0-compatible property.

### Data Type

### String

### See Also

Enabled Property; KeyPress Event.

## DoOption, DontOption Properties

### Applies To

Telnet.

### Description

The DoOption and DontOption properties are set to negotiate with the other end of the connection to enable or disable an option. These properties are not displayed at design time, and are write-only at run time.

### Usage

[*form.*][*control.*]**DoOption** = *numericexpression*

[*form.*][*control.*]**DontOption** = *numericexpression*

### Remarks

When you set the DoOption or DontOption properties, a command is immediately sent across the connection to command the other end to do the option. This tells the other end of the connection, "Do/don't support option x." The other end may respond with either a Will or a Won't command, agreeing or refusing. This command will generate a Cmd event, with the command they sent as an argument.

These properties should only be used when AutoOption is set to False.

For more information on option negotiation, see the Option Negotiation section of the Telnet chapter.

The different options you may send are defined in the POWERTCP.BAS constants file.

### Data Type

### Integer

### See Also

Cmd Event; DoSubOption Property; SubOption Property; WillOption and WontOption Properties.

## DoSubOption Property

### Applies To

Telnet.

### Description

The DoSubOption property sends the option it is set to, plus the sub-option in the SubOption property, to the other end of the connection. This property is not displayed at design time, and is write-only at run time.

### Usage

*[form.]***DoSubOption** = *numericexpression*

### Remarks

The DoSubOption property is used in sub-option negotiation. To send a sub-option, you must follow these steps:

Set the **SubOption** property to the sub-option you want to send. See the **SubOption** property for more information.

Set the **DoSubOption** property to the option you are negotiating. This will send both the option and sub-option to the other end of the connection.

When a control receives a sub-option in the Cmd event, then the Cmd argument will be SB\_CMD, the TelnetOption argument will be equal to the DoSubOption property sent, and the SubOption argument will be identical to the SubOption property sent.

For more information on option negotiation, see the Option Negotiation section of the Telnet chapter.

The different options you can send are defined in the POWERTCP.BAS constants file.

### Data Type

#### Integer

### See Also

Cmd Event; DontOption and DoOption Properties; SubOption Property; WillOption and WontOption Properties.

## **Enabled Property**

### **Applies To**

VT.

### **Description**

Enables the control for display and keyboard input.

### **Usage**

[*form.*][*control.*]**Enabled**[ = {**True**|**False**} ]

### **Remarks**

When set to True, the control will accept display input and will generate KeyDown and KeyPress events when the user types. The cursor is shown if the Cursor property is True.

When set to False, the control will not accept display input or respond to user input. The cursor is then hidden.

### **Data Type**

**Integer** (Boolean)

## **EnableNewLine Property**

### **Applies To**

VT.

### **Description**

Sets or returns a value that determines whether the NewLine event is enabled.

### **Usage**

[*form.*][*control.*]**EnableNewLine**[ = {**True**|**False**} ]

### **Data Type**

**Integer** (Boolean)

### **See Also**

NewLine Event.

## **Enterprise Property**

### **Applies To**

SNMP.

### **Description**

The enterprise name to use for traps. Value is set each time a RecvTrap Event is called. This value is used when sending traps.

### **Usage**

[*form.*][*control.*]**Enterprise**[ = *stringexpression*]

### **Remarks**

The Enterprise property is equal to the sending agent's sysObjectID.

### **Data Type**

**String**



## **ErrorIndex Property**

### **Applies To**

SNMP.

### **Description**

Specifies the index to the variable in error within the packet.

### **Usage**

[*form.*][*control.*]**ErrorIndex**[ = *numericexpression*]

### **Data Type**

### **Integer**

### **See Also**

ErrorStatus Property.

## ErrorStatus Property

### Applies To

SNMP.

### Description

Indicates the type of error associated with the ErrorIndex property.

### Usage

[*form.*][*control.*]**ErrorStatus**[ = *numericexpression*]

### Remarks

The ErrorStatus property may be one of the following:

<b>Value</b>	<b>Name</b>	<b>Description</b>
0	noError	No error exists
1	tooBig	The agent was not able to fit the reply into a single SNMP message
2	noSuchName	The operation specified a variable which did not exist
3	badValue	A set operation specified an invalid value or syntax
4	readOnly	The manager attempted to change a read-only variable
5	genErr	An error occurred which did not fit the other errors

### Data Type

**Integer** (Enumerated)

### See Also

ErrorIndex Property.

## **Expand Property**

### **Applies To**

SMTP.

### **Description**

Set Expand to a mailing list or alias string to have the mail host provide you with a list of names and addresses. This property is write-only and is not displayed at design time.

### **Usage**

*[form.][control.]Expand = stringexpression*

### **Remarks**

This command asks the receiver to confirm that the argument identifies a mailing list, and if so, to return the membership of that list. The full name of the users (if known) and the fully specified mailboxes are returned in a multi-line reply. This command has no effect on any of the reverse-path buffer, the forward-path buffer, or the mail data buffer (see RFC 841). PowerTCP buffers up the reply and calls the Smtpt event with the names/addresses. Each line, formatted as "250 -name <address>", is separated by a CR/LF pair. The last line has no CR/LF on the end.

### **Data Type**

### **String**

### **See Also**

Smtpt Event.

## FileStruct Property

### Applies To

FTP.

### Description

Sends a *File Structure* (STRU) command. Specify the file structure to use. This property is not displayed at design time, and is write-only at run time.

### Usage

[*form.*][*control.*] **FileStruct** = *numericexpression*

### Remarks

The table below shows valid settings:

<b>Setting</b>	<b>Description</b>
0	File (no record structure) - Default
1	Record Structure
2	Page Structure

### Data Type

**Integer**

### See Also

Reply Event.

## Flags Property

### Applies To

TCP, Telnet, SMTP, POP3, FTP, UDP, TFTP, SNMP.

### Description

The Flags property allows you to set certain options for the controls.

### Usage

[*form.*][*control.*]**Flags**[ = *flag* ]

### Remarks

The following constants are available to use:

Constant	Description
PT_NOFLAGS = 0	Use this value to specify no flags.
PT_DEBUG = 1	Enables debugging information to be put into the PowerTCP icon at the bottom of the screen. When set, PTCpxx.DBG is created in your default directory (xx is a number between 1 and 99), which contain all received and sent data.
PT_REUSEADDR = 2	This flag forces the socket to be reusable, meaning that more than one socket can bind to a given port number. This is necessary for some protocols, such as FTP.
PT_KEEPAALIVE = 4	Forces the socket to send "keep-alive" packets, verifying the connection is alive during periods of non-use.
PT_SHOW = 8	Shows an icon at the bottom of the screen for each TCP session. This is helpful in monitoring the state of the connection.
PT_TCPNODELAY = 16	Flag used when creating a TCP connection to enable the TCP_NODELAY socket option.
PT_OVERWRITE = 32	When the <b>OverWrite Property</b> is clear the POP3 and TFTP controls are directed to overwrite existing attachment or output files. Setting the <b>OverWrite Property</b> directs the controls to generate a unique attachment or output filename by overlaying a 1 - 99 count on the last two characters of the filename; e.g. "ABCDEF.DOC", if it already exists, will be received as "ABCDE1.DOC". Up to 99 version are allowed, e.g. "ABCD99.DOC". Further attempts to receive this filename will generate an exception.

These flags are defined in POWERTCP.BAS. They can be combined with the `or` statement. For example:

### Example

```
Tcp1.Flags = PT_DEBUG or PT_SHOW
```

### UDP, SNMP, and TFTP Specific

For UDP, SNMP, and TFTP, the only applicable flags are PT\_DEBUG and PT\_SHOW.

### Data Type

### Integer

## FontName Property

### Applies To

VT.

### Description

Sets and returns the font used by the emulator.

### Usage

[*form.*][*control.*]**FontName** [ = *stringexpression*]

### Remarks

If found in the path, the control will load the POWERVT.FON file on initialization, and remove it when terminated. If not found, the control will select a similar font as described below. This file contains two font face names: VT220\_ascii and VT220\_special. These fonts come in many sizes and are used if specified. The ascii font is similar to the VT220's font in appearance, and the special font contains the graphic characters used to draw lines, boxes, etc.

You can allow the user to specify any font, however, and the control will force it to be fixed in width and will choose the closest matching font loaded on the system. Likewise, double-high and double-wide fonts will be selected, but the appearance may not be as polished as using the VT 220 fonts described above. However, monospaced fonts are suggested (Courier, Courier New, FixedSys, and Terminal).

When notified that the host is sending the control to wide-mode (132 column), you may want to specify a smaller font size to fit it onto the screen (use the FontSize property).

### Data Type

### String

### See Also

Cols Property.

## ForeColor Property

### Applies To

VT.

### Description

Sets or returns the color used for alphanumeric and graphic characters.

### Usage

[*form.*][*control.*]**ForeColor** [ = *numericexpression* ]

### Remarks

RGB and system colors are supported. The new color is immediately used.

### Examples

The following are all valid statements:

```
VT1.ForeColor = RGB(255, 0, 0)      ' Red
VT1.ForeColor = QBColor(12)        ' Red
VT1.ForeColor = &H8000000F         ' System button color
                                   ' (normally gray)
```

### Data Type

**Long**

## GeneralTrap Property

### Applies To

SNMP.

### Description

Specifies a general trap to send or receive. Set this before using the Action property.

### Usage

[*form.*][*control.*]**GeneralTrap**[ = *numericexpression*]

### Remarks

GeneralTrap may be one of the following values:

<b>Constant</b>	<b>Name</b>	<b>Description</b>
SNMP_COLD_START	coldStart	The agent is initializing itself.
SNMP_WARM_START	warmStart	The agent is reinitializing itself.
SNMP_LINK_DOWN	linkDown	An interface has gone from the up to the down state.
SNMP_LINK_UP	linkUp	An interface has gone from the down to the up state.
SNMP_AUTHENTICATION_FAILURE	authenticationFailure	A message was received from an SNMP manager with an invalid community.
SNMP_EGP_NEIGHBOR_LOSS	egpNeighborLoss	An EGP peer changed to the down state.
SNMP_ENTERPRISE_SPECIFIC	enterpriseSpecific	Specific to the SNMP implementation.

### Data Type

**Integer** (Enumerated)



## **HeaderDate Property**

### **Applies To**

SMTP.

### **Description**

When set to True, specifies that PowerTCP should add a "Date:" line in the header of subsequent messages sent. The system date is used.

### **Usage**

[*form.*][*control.*]**HeaderDate** [ = **True** | **False** ]

### **Remarks**

Default is True.

### **Data Type**

**Integer** (Boolean)

## **HeaderFrom Property**

### **Applies To**

SMTP.

### **Description**

When set to True, specifies that PowerTCP should add a "From:" line in the header of subsequent messages sent.

### **Usage**

[*form.*][*control.*]**HeaderFrom** [ = **True** | **False** ]

### **Remarks**

Default is TRUE. PowerTCP uses the value of the Sender property to construct the line.

### **Data Type**

**Integer** (Boolean)

## **HeaderSubject Property**

### **Applies To**

SMTP.

### **Description**

When set to a string value, specifies that PowerTCP should add a "Subject:" line in the header of subsequent messages sent.

### **Usage**

[*form.*][*control.*]**HeaderSubject** [ = *stringexpression* ]

### **Remarks**

If "", then no subject is generated.

### **Data Type**

**String**

## **HeaderTo Property**

### **Applies To**

SMTP.

### **Description**

When set to a string value, specifies that PowerTCP should add a "To:" line in the header of subsequent messages sent.

### **Usage**

[*form.*][*control.*]**HeaderTo** [ = **True** | **False** ]

### **Remarks**

Default is TRUE. Uses Recipient in Recipient string for the header line. Uses the rest of the Recipient string for constructing the "CC:" line (if applicable).

### **Data Type**

**Integer** (Boolean)

## Help Property (FTP)

### Applies To

FTP.

### Description

Sends a *Help* (HELP) command. Causes the server to send helpful information regarding its implementation status over the control connection to the user. May take an argument (e.g. SITE, LIST, etc.). This property is not displayed at design time, and is write-only at run time.

### Usage

*[form.]***Help** = *stringexpression*

### Remarks

Set the Help Property to the FTP command you want help on:

### Example

```
FTP1.Help = "LIST"
```

### Data Type

### String

### See Also

Reply Event.

## Help Property (SMTP)

### Applies To

SMTP.

### Description

Set the Help property to the SMTP message of interest to have the HELP message sent. This property is write-only and is not displayed at design time.

### Usage

*[form.][control.]Help = numericexpression*

### Remarks

This command causes the receiver to send helpful information to the sender of the HELP command. The command takes an argument (e.g., any command name) and returns more specific information as a response. This command has no effect on the reverse-path buffer, the forward-path buffer, or the mail data buffer.

### Example

```
SMTP1.Help = SMTP_MAIL
```

### Data Type

**Integer**

## Keypad Property

### Applies To

VT.

### Description

Sets or returns whether the keypad sends ASCII character codes or escape sequences.

### Usage

[*form.*][*control.*]**Keypad**[= setting%]

<b>Setting</b>	<b>Description</b>
0	Numeric Keypad. Causes the auxiliary keypad to send ASCII character codes corresponding to the numeric characters on the keys.
1	Application Keypad. Causes the auxiliary keypad to send escape sequences used by an application program.

### Data Type

**Integer** (Enumerated)

## **LastCommand Property**

### **Applies To**

FTP.

### **Description**

While waiting for an FTP reply, returns the outstanding FTP command last sent to the FTP server.

### **Usage**

[*form.*][*control.*]**LastCommand**

### **Remarks**

After the FTP reply is received, returns FTP\_CLEAR. This property is not displayed at design time, and is read-only at run-time.

### **Data Type**

**Integer**



## **Lines Property**

### **Applies To**

POP3.

### **Description**

The Lines property should be set before the TopLines property. It specifies the number of lines the POP3 server should provide for the mail message requested. Defaults to 10.

### **Usage**

[*form.*][*control.*]**Lines** = [ *numericexpression* ]

### **Remarks**

See remarks section of TopLines property.

### **Data Type**

#### **Integer**

### **See Also**

TopLines Property.

## List Property (FTP)

### Applies To

FTP.

### Description

Sends a *List* (LIST) command. Causes a list to be sent from the server to the client.

### Usage

[*form.*][*control.*]**List** = *stringexpression (PathName)*

### Remarks

If the pathname specifies a directory or other group of files, the server should transfer a list of files in the specified directory. If the pathname specifies a file then the server should send current information on the file. A null ("" ) argument implies the user's current working or default directory. This property is not displayed at design time, and is write-only at run time.

The following example will cause a listing of root's directory to arrive at the Recv Event:

### Example

```
FTP1.List = "/users/root"
```

### Data Type

**String**

### See Also

Reply Event.

## List Property (POP3)

### Applies To

POP3.

### Description

The List property, when set to 0 or a message number, provides a "scan listing" of all messages or the message referenced. This property is write-only at run-time.

### Usage

[*form.*][*control.*]**List**[ = *numericexpression* ]

### Remarks

If a non-zero argument is given and the POP3 server issues a positive response, then the Pop3 event is called with a line containing information for that message. This line is called a "scan listing" for that message and looks like this:

```
+OK 2 200
```

(MessageNumber is 2, it contains 200 octets, and the string is terminated with a CR/LF)

If a zero argument was given and the POP3 server issues a positive response, then the response is multi-line. After the initial +OK, for each message in the maildrop, the POP3 server responds with a line containing information for that message. This line is also called a "scan listing" for that message. The multi-line looks like this:

```
+OK 2 messages (320 octets)
1 120
2 200
```

In order to simplify parsing, all POP3 servers are required to use a certain format for scan listings. A scan listing consists of the message-number of the message, followed by a single space and the exact size of the message in octets, and a CR/LF. Messages marked as deleted are not listed.

### Data Type

#### Integer

#### See Also

Pop3 Event.

## **ListenTimeout Property (FTP)**

### **Applies To**

FTP.

### **Description**

The ListenTimeout property, when set, will cause the passive data connection to time-out if no connection is established within the time specified. The default is 15 seconds.

### **Usage**

[*form.*][*control.*]**ListenTimeout**[ = *numericexpression* ]

### **Data Type**

**Integer**

## **LocalDotAddr Property**

### **Applies To**

TCP, Telnet, FTP, SMTP, POP3, UDP, SNMP, TFTP.

### **Description**

Used to specify a local host IP address when the default address is not desired. This property is only useful for multi-homed hosts (hosts with more than one network card or network connection) where binding to a specific address is desired.

### **Usage**

[*form.*][*control.*]**LocalDotAddr**[ = *stringexpression*]

### **Remarks**

LocalDotAddr is used to specify the local address used when active and passive connections are requested. This parameter is only necessary on a multi-homed host with more than one IP address. Normally, one does not specify a LocalDotAddr (0 is used), allowing the default address to be used.

### **Data Type**

### **String**

### **See Also**

LocalName Property.

## LocalFileSpec Property

### Applies To

FTP, TFTP.

### Description

Specifies a local filename (with path) before initiating a file transfer.

### Usage

[*form.*][*control.*]**LocalFileSpec**[ = *stringexpression*]

### Remarks

The LocalFileSpec must contain a full path name, with the full filename, such as "c:\temp\test.txt".

**FTP Specific.** For sending files using FTP, the LocalFileSpec should be set to the filename which will be sent (before using the Store, StoreUnique, or Appe properties). For receiving files, the LocalFileSpec should be set to the filename in which the incoming data will be placed (before using the Retrieve property). The Transfer event is called each time a block of data is sent or received, so that your program can monitor the amount of data transferred.

If LocalFileSpec is set to nothing (""), then sending and receiving data must be handled by your code, using the Send and Recv events.

**TFTP Specific.** For TFTP, the LocalFileSpec property must be set before setting the Action property to PUT\_NETASCII, PUT\_OCTET, GET\_NETASCII or GET\_OCTET.

### Data Type

### String

### See Also

Appe Property; Recv Event; Send Event; Store Property; StoreUnique Property.

## **LocalName Property**

### **Applies To**

TCP, Telnet, SMTP, POP3.

### **Description**

Contains the name of the local computer. This property is not available at design time, and read-only at run time.

### **Usage**

[*form.*][*control.*]**LocalName**

### **Remarks**

This property contains the name of your computer **only** while the control is either listening or connected.

### **Data Type**

**String**

### **See Also**

LocalDotAddr Property.

## LocalPort Property

### Applies To

TCP, Telnet, SMTP, POP3, UDP, SNMP.

### Description

Specifies the port to listen on, or the local port to use when connecting.

### Usage

[*form.*][*control.*]**LocalPort**[ = *numericexpression*]

### Remarks

For active connections, setting LocalPort to 0 will cause the protocol to assign an "ephemeral" port address that is subsequently reported by the Connect event. This is the recommended value for active connections. Setting LocalPort to a non-zero will force the socket to bind to that local port, which could fail if that port was used recently (see the flag PT\_REUSEADDR in the Flags property). For passive connections, LocalPort must be set to a value between 1 and 65535.

### Data Type

### Long

### See Also

Action Property (TCP, Telnet); Action Property (SNMP); Action Property (UDP); Flags Property; Listen Event; LocalDotAddr Property.



## LoginHost Property

### Applies To

FTP.

### Description

This property will resolve the specified host name to an IP address , make a control connection, and log into the host.

### Usage

*[form.][control.]LoginHost = stringexpression (RemoteHost)*

### Remarks

If successful, Connect Event is called to notify you of the connection. It will then proceed with logging into the host using the User, Password, and Account Properties specified. Reply Event is fired during this process, informing you of the status of the login process. This property is not displayed at design time, and is write-only at run time. The following example illustrates the process:

```
Sub Form_Load ()
    ' Set remote login information
    FTP1.User = "anonymous" ' for anonymous FTP login
    FTP1.Password = "baldwin@dart.com"
    ' Account property does not usually need to be set
    FTP.LoginHost = "ftp.dart.com" ' our FTP server
    ' connection process starts...
    ' wait for successful connection and login...
    ' Connect Event will be called first...
    ' Reply Event will be called with server replies...
End Sub
```

### Data Type

**String**

### See Also

**Reply Event.**

## **Logout Property**

### **Applies To**

FTP.

### **Description**

Sends a *Logout* (QUIT) command. Use this property to terminate an FTP session.

### **Usage**

[*form.*][*control.*]**Logout = True**

### **Remarks**

If a file transfer is not in progress, the server closes the control connection. If a file transfer is in progress, the connection will remain open for result response and the server will then close it. This property is not displayed at design time, and is write-only at run time.

Logout is normally used to allow the FTP server to terminate service gracefully. If not successful, the CloseControl property can be used to terminate the control connection from the client side.

### **Data Type**

**Integer** (Boolean)

## **MailFileSpec Property**

### **Applies To**

POP3.

### **Description**

Specifies the FileSpec to be used for the next retrieved mail message. May be set before the Retrieve Property is used.

### **Usage**

[*form.*][*control.*]**MailFileSpec** [ = *stringexpression* ]

**Remarks** Received mail is always reported in the Pop3 Event. If this property is set to a valid file specification, then the mail message is also spooled to the file (encoded file attachments are spooled to the directory specified by the AttachmentDir Property).

### **Data Type**

**String**

## **MakeDir Property**

### **Applies To**

FTP.

### **Description**

Sends a *Make Directory* (MKD) command. This causes the directory specified in the pathname to be created. This property is not displayed at design time, and is write-only at run time.

### **Usage**

[*form.*][*control.*]**MakeDir** = *stringexpression (PathName)*

### **Example**

The following example illustrates the creation of a directory:

```
FTP1.MakeDir = "/users/crazyhorse/temp"
```

### **Data Type**

**String**

### **See Also**

Reply Event.

## **MaxRetries Property**

### **Applies To**

TFTP.

### **Description**

The MaxRetries property is used to set the maximum number of times an unacknowledged packet will be re-sent before aborting.

### **Usage**

[*form.*][*control.*]**MaxRetries**[ = *numericexpression* ]

### **Remarks**

The default value is 3. Minimum is 1 and maximum is 5.

### **Data Type**

**Integer**

### **See Also**

TimeoutInterval Property.

## **Message Property**

### **Applies To**

SMTP.

### **Description**

Mail message to be sent when the Action property is used to send mail.

### **Usage**

[*form.*][*control.*]**Message**[ = *stringexpression* ]

### **Remarks**

This is a level 1 string (NULL terminated with no embedded NULLs).

### **Data Type**

**String**

## Mode Property

### Applies To

FTP.

### Description

Sends a *Transfer Mode* (MODE) command. Specifies the data transfer mode. This property is not displayed at design time, and is write-only at run time.

### Usage

[*form.*][*control.*]**Mode** = *numericexpression*

### Remarks

The table below lists the valid modes:

<b>Setting</b>	<b>Description</b>
0	Stream Transfer Mode (Default)
1	Block Transfer Mode
2	Compressed Transfer Mode

### Data Type

**Integer**

### See Also

Reply Event.

## **NameList Property**

### **Applies To**

FTP.

### **Description**

Sends a *Name List* (NLST) command. This causes a directory listing to be sent from server to user site.

### **Usage**

[*form.*][*control.*]**NameList** = *stringexpression (PathName)*

### **Remarks**

The pathname should specify a directory or other system-specific file group descriptor; a NULL argument implies the current directory. The server will return a stream of names of files in the Recv event and no other information. This property is not displayed at design time, and is write-only at run time. The following example illustrates the listing of a directory:

### **Example**

```
FTP1.NameList = "/users/root"
```

### **Data Type**

**String**

### **See Also**

Reply Event.



## **NewLine Property**

### **Applies To**

VT.

### **Description**

Sets or returns the selected end-of-line code.

### **Usage**

[*form.*][*control.*]**NewLine**[= setting]

### **Remarks**

Some hosts would prefer a CR/LF pair as an end-of-line code, sent when the Enter key is pressed.

<b>Setting</b>	<b>Description</b>
----------------	--------------------

---

0	CR only (Chr\$(13))
1	CR/LF pair (Chr\$(13) & Chr\$(10))

### **Data Type**

**Integer** (Enumerated)

## **nObjects Property**

### **Applies To**

SNMP.

### **Description**

Specifies number of objects contained in the ObjectID, ObjectType, and ObjectValue arrays.

### **Usage**

[*form.*][*control.*]**nObjects**[ = *numericexpression*]

### **Remarks**

Since these arrays are dynamically allocated, the using program must set the nObjects property to the correct size *before* assigning values to these arrays. When read from within the RecvSnmP and RecvTrap events, specifies the number of objects received within that Datagram packet.

### **Data Type**

**Integer**

## **NodeDotAddr Property**

### **Applies To**

SNMP.

### **Description**

Specifies the IP address (x.x.x.x) of the node used when a Trap message is sent (using the Action property) or received (in the Trap Event).

### **Usage**

[*form.*][*control.*]**NodeDotAddr**[ = *stringexpression*]

### **Data Type**

**String**

## **Noop Property**

### **Applies To**

FTP.

### **Description**

Sends a *NoOp* (NOOP) command. Specifies no action other than that the server send an OK reply. This property is not displayed at design time, and is write-only at run time.

### **Usage**

[*form.*][*control.*]**Noop = True**

### **Data Type**

**Integer** (Boolean)

### **See Also**

Reply Event.

## ObjectID Property

### Applies To

SNMP.

### Description

Specifies an array of strings where each entry specifies an SNMP object such as "1.3.6.1.2.1.1.1.0" for sysDescr. This property is not available at design-time, and is read/write at run-time.

### Usage

[*form.*][*control.*]**ObjectID**(*index*)[ = *stringexpression*]

### Remarks

PowerTCP updates this array before RecvSnmpp or RecvTrap events are called. For every entry in the ObjectID property array, there is a corresponding entry in the ObjectValue and ObjectType property arrays.

The nObjects property is used to determine or set the size of the array.

### Data Type

**String**

### See Also

ObjectValue Property; ObjectType Property.

## ObjectValue Property

### Applies To

SNMP.

### Description

An array of strings where each entry specifies an SNMP object value. This property is not available at design-time, and is read/write at run-time.

### Usage

[*form.*][*control.*]**ObjectValue**(*index*)[ = *stringexpression*]

### Remarks

PowerTCP updates this array before RecvSnmp and RecvTrap events are called. For every entry in the ObjectID property array, there is a corresponding entry in the ObjectValue and ObjectType property arrays.

The nObjects property is used to determine or set the size of the array.

### Data Type

**String**

### See Also

ObjectID Property; ObjectType Property.

## ObjectType Property

### Applies To

SNMP.

### Description

An array of integers where each entry specifies an SNMP object type. This property is not available at design-time, and is read/write at run-time.

### Usage

[*form.*][*control.*]**ObjectType(n)**[ = *numericexpression*]

### Remarks

PowerTCP updates this array before RecvSnmp and RecvTrap events are called. ObjectType specifies an SMI type(basically either a Long or a String). For every entry in the ObjectID property array, there is a corresponding entry in the ObjectValue and ObjectType property arrays.

The nObjects property is used to determine or set the size of the array.

Possible settings include:

### Constant

---

SNMP\_INTEGER = 2

SNMP\_OCTET\_STRING = 4

SNMP\_NULL = 5

SNMP\_OBJECT\_ID = 6

SNMP\_SEQUENCE = 48

SNMP\_SET\_OF = 49

SNMP\_IPADDRESS = 64

SNMP\_COUNTERS = 65

SNMP\_GUAGE32 = 66

SNMP\_TIMETICKS = 67

SNMP\_OPAQUE = 68

SNMP\_NSAP = 69

SNMP\_COUNTER64 = 70

SNMP\_UINTEGER32 = 71

### Data Type

#### Integer

#### See Also

ObjectID Property; ObjectValue Property.

## **OemLicense Property**

### **Applies To**

TCP, Telnet, FTP, SMTP, POP3, VT, UDP, SNMP, TFTP.

### **Description**

Specifies a unique license string given to you by Dart Communications.

### **Usage**

[*form.*][*control.*]**OemLicense**[ = *stringexpression*]

### **Remarks**

You must set the OemLicense to its correct value at run-time before the Action property is set. This property is used to provide optimal performance for OEM customers.

### **Data Type**

**String**



## Passive Property

### Applies To

FTP.

### Description

Sends a *Passive* (PASV) command. Requests the FTP server listen on a data port. This property is not displayed at design time, and is write-only at run time.

### Usage

[*form.*][*control.*]**Passive** = **True**

### Data Type

**Integer** (Boolean)

### See Also

Reply Event.

## **Password Property (FTP)**

### **Applies To**

FTP.

### **Description**

Specifies the account password used when logging in.

### **Usage**

[*form.*][*control.*]**Password** = *stringexpression (user password)*

### **Remarks**

Must be set before using the LoginHost property.

### **Data Type**

**String**

### **See Also**

Account Property, User Property.

## **Password Property (POP3)**

### **Applies To**

POP3.

### **Description**

Specifies the password to be used for authentication. Must be set prior to setting Action=CONNECTCOMM.

### **Usage**

[*form.*][*control.*]**Password** = *stringexpression*

### **Remarks**

USER/PASS authentication is used unless the Secret property is set, indicating that APOP authentication is to be used.

### **Data Type**

**String**

## Port Property

### Applies To

FTP.

### Description

Sends a *Data Port* (PORT) command. Use this property to tell the FTP server what host address and port it should use for the next FTP data connection. This property is not displayed at design time, and is write-only at run time.

### Usage

*[form.]***Port** = *stringexpression*

### Remarks

For controlling two FTP servers and transferring files between them, use the Passive property to instruct one server to accept the data connection, then use the Port Property to instruct the second what host and port it should connect to.

For sending and receiving files and lists, PowerTCP automatically sends the PORT command so you do not have to use this property for normal operations.

If desired, however, a local port can be opened explicitly for commands sent using the Command property. Set Port equal to "". This will cause the library to open a listening socket, construct a PORT command, and send it over the control connection. The subsequent file transfer command will cause the server to connect to the specified port.

### Data Type

**String**

### See Also

Reply Event.

## **PrinterController Property**

### **Applies To**

VT.

### **Description**

When set to True, disables the display of text and sends all data to the default printer. When set to False, the Display Property displays text normally and nothing is sent to the printer.

### **Usage**

```
[form.][control.]PrinterController[ = {True|False} ]
```

### **Remarks**

Behavior is the same as the VT220.

### **Data Type**

**Integer** (Boolean)

## **PrintPassthrough Property**

### **Applies To**

VT.

### **Description**

When set to True, normal printer drivers are bypassed and characters proceed directly to the printer port.

### **Usage**

[*form.*][*control.*]**PrintPassthrough**[ = {**True|False**} ]

### **Remarks**

This is used in conjunction with the [AutoPrint](#), [PrinterController](#) and [PrintScreen](#) properties. PrinterController must be set before entering PrinterController or AutoPrint mode.

### **Data Type**

**Integer** (Boolean)

## **PrintScreen Property**

### **Applies To**

VT.

### **Description**

When set to True, sends a printout of the current screen (normally 24x80) to the printer. This is write-only at run-time.

### **Usage**

`[form.][control.]PrintScreen = True`

### **Data Type**

**Integer** (Boolean)

## **PrintWorkingDir Property**

### **Applies To**

FTP.

### **Description**

Sends a *Print Working Directory* (PWD) command. Causes the name of the current working directory to be returned in the reply. This property is not displayed at design time, and is write-only at run time.

### **Usage**

[*form.*][*control.*]**PrintWorkingDir = True**

### **Data Type**

**Integer** (Boolean)

### **See Also**

Reply Event.



## **Recipients Property**

### **Applies To**

SMTP.

### **Description**

Specifies the recipients of subsequent mail messages as a comma-delimited string of addresses. Write-only at run-time.

### **Usage**

[*form.*][*control.*]**Recipients** = *stringexpression*

### **Remarks**

The Recipient string should be used to specify the mail recipient(s) before the Action property is set.

### **Data Type**

**String**

## **Recv Property**

### **Applies To**

TCP, Telnet.

### **Description**

The Recv property allows you to set the maximum size of received strings reported by the Recv Event.

### **Usage**

*[form.][control.]Recv = numericexpression*

### **Remarks**

Recv can be set to 0 to disable the reporting of data, providing backpressure over the TCP connection. When called, the receive buffers are immediately checked for data, and if present, will be reported with one call to the Recv Event.

The Recv Property should be used in special cases only, and only while a connection exists. The Recv Property has no effect when a connection does not exist. The Recv Property must be set each time a connection is made, as the value is not persistent from one connection to the next.

### **Data Type**

**Integer**

## Reinitialize Property

### Applies To

FTP.

### Description

Sends a *Reinitialize* (REIN) command. Use this property to terminate a USER, flushing all I/O and account information, except to allow the transfer in progress to complete. This property is not displayed at design time, and is write-only at run time.

### Usage

[*form.*][*control.*]**Reinitialize = True**

### Data Type

**Integer** (Boolean)

### See Also

Reply Event.

## **RemoteFileSpec Property**

### **Applies To**

TFTP.

### **Description**

Specifies a remote file name (with path) before setting Action = PUT\_NETASCII, PUT\_OCTET, GET\_NETASCII or GET\_OCTET.

### **Usage**

[*form.*][*control.*]**RemoteFileSpec**[ = *stringexpression*]

### **Remarks**

Specifies the remote file used for getting data from, or the remote file used for sending data to during file transfers.

### **Data Type**

**String**

## RemoteHost Property

### Applies To

TCP, Telnet, SMTP, POP3, UDP, SNMP, TFTP.

### Description

Specifies the name or IP address (x.x.x.x) of a remote computer to connect to, or that you are currently connected to. Or, for UDP-based protocols, specifies the destination of packets.

### Usage

[*form.*][*control.*]**RemoteHost**[ = *stringexpression*]

### Remarks

Before connecting to a remote computer, you must specify the RemoteHost and RemotePort properties. RemoteHost identifies the computer to connect to, and RemotePort identifies the port to use.

After you set the RemoteHost and RemotePort properties, you can connect to a remote computer by setting the Action property to CONNECTCOMM.

The RemoteHost property can also be used to determine what computer you are currently connected to. If you already accepted a connection from a remote computer by using the Session property, you can check what computer you are connected to by reading the RemoteHost property.

### Data Type

#### String

### See Also

Action Property (TCP, Telnet); Action Property (SMTP); Action Property (POP3); Action Property (TFTP); Action Property (UDP); Action Property (SNMP); RemotePort Property.

## RemotePort Property

### Applies To

TCP, Telnet, SMTP, POP3, SNMP.

### Description

Specifies the port to make a connection to.

### Usage

[*form.*][*control.*]**RemotePort**[ = *numericexpression*]

### Remarks

This property must be set before a connection can be made. Different protocols have different standard values for this. The table below lists the standard RemotePort for several protocols:

<b>Protocol</b>	<b>Port</b>
ECHO	7
DISCARD	9
DAYTIME	17
CHARGEN	19
Telnet	23
FTP	21
SMTP	25
POP3	110
SNMP Agent	161 (UDP)
SNMP Trap	162 (UDP)
REXEC	512
RLOGIN	513
RSH	514

### Data Type

**Long**

### See Also

Action Property (TCP, Telnet); Action Property (SNMP); RemoteHost Property.

## RemoveDir Property

### Applies To

FTP.

### Description

Sends a *Remove Directory* (RMD) command. Causes the directory specified to be removed. This property is not displayed at design time, and is write-only at run time.

### Usage

[*form.*][*control.*] **RemoveDir** = *stringexpression* (*PathName*)

### Remarks

To remove a directory:

### Example

```
FTP1.RemoveDir = "/user/root/test"
```

### Data Type

### String

### See Also

Reply Event.

## Rename Property

### Applies To

FTP.

### Description

When used, this property first sends a *Rename From* (RNFR) command, waits for a reply, then sends a *Rename To* (RNTO) command and waits for a reply. The Reply Event uses FTP\_WORKING for Status when reporting the intermediate step. This property is not displayed at design time, and is write-only at run time.

### Usage

[*form.*][*control.*] **Rename** = *stringexpression*

### Remarks

To rename a file specify both files on the command line with a space in between:

### Example

```
FTP1.Rename = "/user/root/from.txt /user/root/to.txt"
```

### Data Type

### String

### See Also

Reply Event.



## **RequestID Property**

### **Applies To**

SNMP.

### **Description**

Unique number used to distinguish outstanding packets. This is the best tool to determine what response belongs to which requests when they are returned in the RecvSnmp Event.

### **Usage**

[*form.*][*control.*]**RequestID**[ = *numericexpression*]

### **Data Type**

**Long**

### **See Also**

Action Property.

## **Reset Property**

### **Applies To**

VT.

### **Description**

Resets the display characteristics. Write-only at runtime.

### **Usage**

[*form.*][*control.*]**Reset = True**

### **Remarks**

When set to True, resets display attributes (cursor position to home, etc.). This does not clear the screen. Use the Clear method to fill the display with blanks.

### **Data Type**

**Integer** (Boolean)

## **Restart Property**

### **Applies To**

FTP.

### **Description**

Sends a *Restart* (REST) command. Restarts file transfer from the marker specified. This property is not displayed at design time, and is write-only at run time.

### **Usage**

[*form.*][*control.*] **Restart** = *stringexpression*

### **Remarks**

Set the Restart Property to the value of the server marker (data checkpoint).

### **Data Type**

**String**

### **See Also**

Reply Event.

## Retrieve Property (FTP)

### Applies To

FTP.

### Description

Sends a *Retrieve* (RETR) command. Causes the server to make a data connection and transfer a copy of the file specified to the client. This property is not displayed at design time, and is write-only at run time.

### Usage

[*form.*][*control.*] **Retrieve** = *stringexpression (PathName)*

### Remarks

The Recv Event is called with data sent by the server. If the LocalFileSpec property is specified (not equal to ""), then PowerTCP will take care of placing incoming data into the file.

### Data Type

### String

### See Also

Recv Event, Reply Event.

## Retrieve Property (POP3)

### Applies To

POP3.

### Description

When set to a message number, instructs the POP3 server to send the specified message. This property is not displayed at design time, and write-only at run time.

### Usage

[*form.*][*control.*]**Retrieve** = *numericexpression*

### Remarks

The AttachmentDir and MailFileSpec properties should be set prior to setting this property, if desired.

If the POP3 server issues a positive response, then the response given is multi-line. After the initial +OK, the POP3 server sends the message corresponding to the given message-number. The Pop3 Event will likely be called numerous times for large mail messages.

If the POP3 server issues a positive response, then the response given is multi-line. After the initial +OK, the POP3 server sends the message corresponding to the given message-number. This message may include files encoded using MIME or UUENCODE techniques. For large messages, Pop3 Event will likely be called numerous time.

The mail message and attached file data is provided in the *Reply* parameter of Pop3 Event. An Pop3 Event call with *status* = **POP3\_HEADER** signals the complete mail header. This will be followed by a Header Event which returns the parsed common header fields.

Calls to Pop3 Event with *status* = **POP3\_MAIL** will return the message data. The mail body is only saved to disk if *MailFileSpec* is not NULL.

Attachments are only decoded if a valid *AttachmentDir* is specified. Calls to Pop3 Event with *status* = **POP3\_FILE** will return the decoded attachment data. File Event signals the closing of an attachment file, and the attachment file name actually created.

### Data Type

**Integer**

### See Also

AttachmentDir Property; MailFileSpec Property.

## **RowHeight Property**

### **Applies To**

VT.

### **Description**

Returns the height of a single character (or line) according to the measurement units set up by the container object (normally the form).

### **Usage**

[*form.*][*control.*]**RowHeight**

### **Remarks**

Read-only at runtime. When multiplied by the Rows property, the application can compute the height of the control necessary to view one screen of data.

### **Data Type**

**Integer**

### **See Also**

Rows Property.

## **Rows Property**

### **Applies To**

VT.

### **Description**

Sets or returns the number of rows being used for the screen.

### **Usage**

[*form.*][*control.*]**Rows**[=*setting*]

### **Remarks**

Normally 24, but some programs support larger numbers. This can be dynamically re-sized, with PowerTCP preserving as much data as possible.

### **Data Type**

**Integer**

### **See Also**

Cols Property.

## Scroll Property

### Applies To

VT.

### Description

Sets or returns how quickly lines appear on the screen.

### Usage

[*form.*][*control.*]**Scroll**[= setting%]

<b>Setting</b>	<b>Description</b>
0	Jump Scroll. Displays new lines as fast as they are received, causing a jump scroll.
1	Smooth Scroll. Limits the speed at which new lines appear on the screen, causing a smooth and steady scroll.

### Data Type

**Integer** (Enumerated)



## **Secret Property**

### **Applies To**

POP3.

### **Description**

This property is used to set the "shared secret" that the APOP message uses to minimize username/password traffic on the network. Not currently implemented.

### **Usage**

[*form.*][*control.*]**Secret** = *stringexpression*

### **Remarks**

If set to "", normal USER/PASS messages are used for authentication purposes. If set to the "shared secret" string that is also known to the POP3 server, then an APOP message is sent. Refer to the POP3 RFC for a complete description.

### **Data Type**

**String**

## **SelLength Property**

### **Applies To**

VT.

### **Description**

Sets or returns the length of the text area highlighted by the user on the screen.

### **Usage**

[*form.*][*control.*]**SelLength**[=setting%]

### **Remarks**

When set to 0, no text is selected.

### **Data Type**

**Integer**

### **See Also**

SelStart Property; Text Property.

## **SelStart Property**

### **Applies To**

VT.

### **Description**

Sets or returns the start of the text area highlighted by the user on the screen.

### **Usage**

```
[form.][control.]SelStart [ = setting% ]
```

### **Remarks**

The user is allowed to highlight text anywhere on the main screen or in the scrollbar buffer area. SelStart provides a 0-based index into the Text array (the first character starts at 0) to allow the program access to the highlighted text. Use the InStr function to access selected text.

To determine what area of the display is selected, multiply the number of columns by the number of rows in the scrollbar buffer (BufferRows property). This provides an offset into the display area, from which any screen location can be calculated.

### **Example**

To access the currently selected text, use the following line, where *VT1* is the name of the control:

```
Mid$(VT1.Text, VT1.SelStart, VT1.SelLength)
```

### **Data Type**

Integer

### **See Also**

SelLength Property; Text Property, Style Property.

## Send Property

### Applies To

TCP, Telnet, FTP, UDP.

### Description

Sends a string of text to a remote computer. This property is not displayed at design time, and write-only at run time.

### Usage

[*form.*][*control.*]**Send** = *stringexpression*

### Remarks

To send data the State property must be CONNECTED. To send a message to the other end of the connection, assign a string of text to the Send property. This sends the text to the other computer.

When the string is accepted by the local system for transmission the Send Event is fired. The remote computer will receive a Recv event, with the sent text as an argument.

The Send property *may* include null characters, as it is a Visual Basic 2.0 property. For environments which use null-terminated strings (such as Delphi, PowerBuilder, and Microsoft Visual C++), the SendString property should be used.

### FTP Specific

When used in the FTP control, this sends data across the data connection, and not the control connection. When a data connection is first established, PowerTCP fires the Send Event to signal the application it can start processing the FTP STOR, STOU, or APPE operation. When finished sending data, the CloseData Property should be used to signal PowerTCP to close the data connection.

### UDP Specific

UDP is a Visual Basic 2.0 control, and so does not support the SendByte or SendString properties.

### Data Type

#### String

### See Also

Recv Event; Send Event; DataTag Property; SendByte Property, SendString Property; Urgent Property.

## **SendByte Property**

### **Applies To**

TCP, Telnet, FTP.

### **Description**

Sends a byte to a remote computer. This property is not displayed at design time, and write-only at run time.

### **Usage**

*[form.]***SendByte** = *numericexpression*

### **Remarks**

This property is similar to the Send Property, but sends a single byte. This is useful for sending a NULL character in a programming environment that only supports Visual Basic 1.0 custom controls. Values may range from 0 to 255.

### **Data Type**

#### **Integer**

### **See Also**

Send Property; Send Event; DataTag Property.

## **Sender Property**

### **Applies To**

SMTP.

### **Description**

Used to specify the address of the sender of the mail message.

### **Usage**

[*form.*][*control.*]**Sender** = *stringexpression*

### **Remarks**

Must be set to a proper value prior to sending mail.

### **Data Type**

**String**

## **SendString Property**

### **Applies To**

TCP, Telnet, FTP.

### **Description**

Sends a null-terminated string to a remote computer (the null character is not sent). This property is not displayed at design time, and write-only at run time.

### **Usage**

[*form.*][*control.*]**SendString** = *stringexpression*

### **Remarks**

This property is similar to the Send property, but uses null-terminated strings. This property should not be used in Visual Basic, but must be used in Delphi, PowerBuilder, and Microsoft Visual C++, as these environments use null-terminated strings. Note that this is a Visual Basic 1.0-compatible property.

### **Data Type**

### **String**

### **See Also**

Recv Event; Send Event; DataTag Property; SendByte Property, SendString Property; Urgent Property.

## **ServerPort Property**

### **Applies To**

TFTP.

### **Description**

When functioning as a server, this property allows you to set the server port to a different value than the "well-known" TFTP port of 69. When functioning as a client, this property allows you to contact a TFTP server that is using a port other than port 69.

### **Usage**

[*form.*][*control.*]**ServerPort**[ = *numericexpression* ]

### **Remarks**

Default value is 69.

### **Data Type**

**Long**



## Session Property

### Applies To

TCP, Telnet.

### Description

The Session property accepts a connection sent to a control in Listen mode. This property is not available at design time, and write-only at run time.

### Usage

[*form.*][*control.*]**Session** = *numericexpression*

### Remarks

When a remote computer connects to a "listening" control, the listening control must "pass on" the connection to another PowerTCP control. To do this, you must perform the following steps:

- Find the NewSession parameter in the Accept event of the listening control (this represents information about the connection).
- Assign the NewSession parameter to the Session property of a new control (you may want to use a control array for this, to accept numerous connections). This control now becomes the other end of the connection initiated by the remote computer. A Connect event will fire in the new control, telling the program that the connection was successful.

### Example

A typical use of this property is shown below:

```
Sub TCP1_Accept(NewSession As Long)
    ' Transfer the connection to another TCP control
    TCP2.Session = NewSession
End Sub
```

### Data Type

**Long**

### See Also

Accept Event; Listen Event.

## Site Property

### Applies To

FTP.

### Description

Sends a *Site Parameters* (SITE) command. Used by the server to provide services specific to its system that are essential to file transfer but not sufficiently universal to be included as commands in the protocol. This property is not displayed at design time, and is write-only at run time.

### Usage

[*form.*][*control.*]**Site** = *stringexpression*

### Data Type

**String**

### See Also

Send Event, Reply Event.

## **SpecialTrap Property**

### **Applies To**

SNMP.

### **Description**

Enterprise specific trap. Defined by agent application vendor.

### **Usage**

[*form.*][*control.*]**SpecialTrap**[ = *numericexpression*]

### **Remarks**

Must be set before Action is set to SNMP\_TRAP. Usually set to 0.

### **Data Type**

**Integer**

## State Property

### Applies To

TCP, Telnet, SMTP, POP3, UDP, SNMP, TFTP.

### Description

The State property contains a value representing the current state of the control. This property is not displayed at design time, and read-only at run time.

### Usage

[*form.*][*control.*]**State**

### Remarks

You can read the State property to check on the current state of the control. The state may be one of the following (the constants are defined in POWERTCP.BAS):

<b>Constant</b>	<b>Description</b>
CLOSED = 1	The control has not allocated any resources.
CONNECTING = 2	The control is in the process of trying to connect to a remote computer.
CONNECTED = 4	The control is actively connected to a remote computer, or a UDP-based protocol has allocated resources.
LISTENING = 8	The control is currently listening for new connections.
CLOSING = 16	The control is in the process of closing (outstanding buffers may be flushing).

### UDP, SNMP Specific

The CONNECTING and LISTENING states will not occur in these component s.

### TFTP Specific

The CONNECTING state will not occur in the TFTP component.

### Example

The following example will send a string of data only if a connect exists:

```
If TCP1.State = CONNECTED Then
    TCP1.Send = "How many TCP/IP programmers does it take to screw in a light bulb?"
End If
```

### Data Type

#### Integer

### See Also

Action Property (TCP, Telnet); Action Property (SMTP); Action Property (POP3); Action Property (UDP); Action Property (SNMP); Action Property (TFTP).

## **Status Property**

### **Applies To**

FTP.

### **Description**

Sends a *Status* (STAT) command. Causes a status response to be sent over the control connection in the form of a reply. This property is not displayed at design time, and is write-only at run time.

### **Usage**

[*form.*][*control.*]**Status** = *stringexpression*

### **Data Type**

**String**

### **See Also**

Reply Event.

## Store Property

### Applies To

FTP.

### Description

Sends a *Store* (STOR) command. Causes the server to accept the data transferred via the data connection and to store the data as a file at the server site. This property is not displayed at design time, and is write-only at run time.

### Usage

[*form.*][*control.*]**Store** = *stringexpression (PathName)*

### Remarks

If the LocalFileSpec property is set to nothing (""), then once the data connection is established, PowerTCP calls the Send event to signal the application to begin sending data using the Send property.

If the LocalFileSpec property contains a filename, then PowerTCP sends the data automatically from the file.

The Type property is checked and the TYPE command is sent if the FTP server needs to be informed of a different data type.

### Data Type

#### String

### See Also

Send Property, Send Event, Reply Event.

## StoreUnique Property

### Applies To

FTP.

### Description

Sends a *Store Unique* (STOU) command. Similar to the Store Property, except that the resultant file is to be created in the current directory under a name unique to that directory. This property is not displayed at design time, and is write-only at run time.

### Usage

[*form.*][*control.*]**StoreUnique** = *stringexpression (PathName)*

### Remarks

If the LocalFileSpec property is set to nothing (""), then once the data connection is established, PowerTCP calls the Send event to signal the application to begin sending data using the Send property.

If the LocalFileSpec property contains a filename, then PowerTCP sends the data automatically from the file.

The Type property is checked and the TYPE command is sent if the FTP server needs to be informed of a different data type.

### Data Type

#### String

### See Also

Send Property, Send Event, Reply Event.

## **StructMount Property**

### **Applies To**

FTP.

### **Description**

Sends a *Structure Mount* (SMNT) command. Use this property to mount a different file system data structure without altering login or accounting information. This property is not displayed at design time, and is write-only at run time.

### **Usage**

[*form.*][*control.*]**StructMount** = *stringexpression*

### **Data Type**

**String**

### **See Also**

Reply Event.



## Style Property

### Applies To

VT.

### Description

Contains a string of bytes (stored as characters) which represent style information about their associated character in the Text property. Read-only at runtime.

### Usage

[*form.*][*control.*]**Style**

### Remarks

Each character is a bitwise OR of:

- 0 - Normal
- 1 - Inverse
- 2 - Bold
- 4 - Blink
- 8 - Underline

The Style property is always the same size as the Text property.

### Example

The following example determines if the first character at the top of the buffer in the control is bold or not. The Asc() function converts the character to a number, and it is then combined with 2 using the And function to see if bit 2 is set.

```
If (Asc(Left$(VT.Style, 1)) And 2) Then MsgBox "We have a winner--the character is BOLD!!!"
```

### Data Type

### String

### See Also

Text Property.

## SubOption Property

### Applies To

Telnet.

### Description

The SubOption property is set to send a sub-option along with an option during option negotiation. This property is not displayed at design time, and write-only at run time.

### Usage

[*form.*][*control.*]**SubOption** = *stringexpression*

### Remarks

Setting only the SubOption property does not send the sub-option. See the DoSubOption Property for information on how to send sub-options. This is a Visual Basic 2.0 property, and may include embedded nulls. Sub-option support in Visual Basic 1.0-compatible environments (Delphi, PowerBuilder, Visual C++) can be provided by using the SendString and SendByte properties to send the proper sequence of bytes.

This property should only be used when the AutoOption property is set to False.

### Data Type

**String**

### See Also

Cmd Event; DoOption and DontOption Properties; DoSubOption Property; WillOption, WontOption Properties.

## **System Property**

### **Applies To**

FTP.

### **Description**

Sends a *System* (SYST) command. Used to find out the type of operating system at the server. This property is not displayed at design time, and is write-only at run time.

### **Usage**

[*form.*][*control.*]**System = True**

### **Data Type**

**Integer** (Boolean)

### **See Also**

Reply Event.

## **Tabs Property**

### **Applies To**

VT.

### **Description**

Sets or returns a list of comma delimited tab stops to be used for display formatting.

### **Usage**

[*form.*][*control.*]**Tabs** = *stringexpression*

### **Example**

```
VT1.Tabs = "9, 17, 25, 33, 41, 49, 57, 65, 73"
```

### **Data Type**

**String**

## Terminal Property

### Applies To

VT.

### Description

Sets or returns the terminal type being emulated.

### Usage

[*form.*][*control.*]**Terminal**[= setting%]

<b>Constant</b>	<b>Description</b>
-----------------	--------------------

---

VT\_TTY = 0 TTY. All control sequences are displayed, not interpreted.

VT\_VT52 = 1 VT52 emulation.

VT\_VT100 = 2 VT100 emulation

VT\_VT220\_7 = 3 VT220 emulation with 7-bit controls (default)

VT\_VT220\_8 = 4 VT220 emulation with 8-bit controls

### Data Type

**Integer** (Enumerated)

## Text Property

### Applies To

VT.

### Description

A string of text which contains every character in the buffer and screen sections of the control. Read-only at runtime.

### Usage

[*form.*][*control.*]**Text**

### Remarks

Each line in the Text property is terminated by a CR/LF pair (Chr\$(13) & Chr\$(10)). The Text string can be very large as it contains the scrollbar buffer (number of columns plus 2 for a CR/LF pair, then multiplied by BufferSize) followed by the screen contents (number of columns plus 2, then multiplied by Rows). Use SelStart as a pointer into the start of any selected text, and SelLength for the length of the selected text.

### Example

The following example copies all the text in the current screen (but not buffer) into the clipboard:

```
Clipboard.Clear      ' Clear all contents  
Clipboard.SetText Right$(VT1.Text, (VT1.Rows + VT1.BufferRows) * (VT1.Cols + 2))
```

### Data Type

#### String

### See Also

Style Property; SelStart Property; SelLength Property; Rows Property; Cols Property.

## **TimeStamp Property**

### **Applies To**

SNMP.

### **Description**

Value to send (when Action is used to send a Trap message) or value received (when RecvTrap Event is called).

### **Usage**

[*form.*][*control.*]**TimeStamp**[ = *numericexpression*]

### **Remarks**

This value is in hundredths of seconds from agent startup.

### **Data Type**

**Long**

### **See Also**

Action Property.

## **TimeoutInterval Property**

### **Applies To**

TFTP.

### **Description**

The TimeoutInterval property is used to set the number of seconds PowerTCP will wait for an acknowledgment until re-sending a packet.

### **Usage**

[*form.*][*control.*]**TimeoutInterval**[ = *numericexpression* ]

### **Remarks**

The default value is 1000 (one second).

### **Data Type**

**Integer**

### **See Also**

MaxRetries Property.



## TopLines Property

### Applies To

POP3.

### Description

The TopLines property, when set to a message number, uses the Lines property to construct and send a POP3 TOP message.

### Usage

[*form.*][*control.*]**TopLines**[ = *numericexpression* ]

### Remarks

If the POP3 server issues a positive response, then the reply is multi-line. After the initial +OK, the POP3 server sends the headers of the message, a blank line separates the headers from the body, and then the number of lines in the message's body. Note that if the number of lines requested by the POP3 client is greater than the number of lines in the body, then the POP3 server sends the entire message.

This header and message data is provided to the application through Pop3 Event with *status* = POP3\_MAIL. This will be immediately followed by a Header Event which provides the parsed common header fields. Note that if Header Event fields are to be used, the buffer returned from Pop3 Event with *status* = POP3\_MAIL must not be modified.

### Data Type

**Integer**

### See Also

Lines Property.

## Type, TypeTransfer Properties

### Applies To

FTP.

### Description

Sends a *Representation Type* (TYPE) command. Refer to RFC 959 for a complete description of possible representation types.

### Usage

[*form.*][*control.*]**Type** = *numericexpression*

**Remarks** The Type property is an enumerated type. The TypeTransfer property is an integer, and is compatible with the PowerBuilder environment (Type is not a compatible word in that environment). Type or TypeTransfer should be set before using Appe, Store, StoreUnique or Retrieve.

<b>Constant</b>	<b>Description</b>
-----------------	--------------------

---

FTP_ASCII = 0	Every line is terminated with a CR/LF sequence for transfer
---------------	---

FTP_EBCDIC = 1	EBCDIC transfer.
----------------	------------------

FTP_IMAGE = 2	Exact binary image transferred
---------------	--------------------------------

FTP_IGNORE = 3	Do not enforce any type. Use current state for the transfer.
----------------	--

### Data Type

**Integer (enumerated type for Type Property)**

### See Also

Reply Event.

## Uidl Property

### Applies To

POP3.

### Description

When set to a message number, asks the POP3 server to reply with the unique ID of the message. When set to 0, asks the POP3 server to reply with all unique IDs. This property is not displayed at design time, and read-only at run time. Not currently implemented.

### Usage

*[form.][control.]Uidl = numericexpression*

### Remarks

When set to a message number, the POP3 server issues a positive response with a line containing information for that message. This line is called a "unique-id listing" for that message.

When set to 0, and the POP3 server issues a positive response, then the response given is multi-line. After the initial +OK, for each message in the maildrop, the POP3 server responds with a line containing information for that message. This line is called a "unique-id listing" for that message.

In order to simplify parsing, all POP3 servers are required to use a certain format for unique-id listings. A unique-id listing consists of the message-number of the message, followed by a single space and the unique-id of the message. No information follows the unique-id in the unique-id listing.

The unique-id of a message is an arbitrary server-determined string, consisting of characters in the range 0x21 to 0x7E, which uniquely identifies a message within a maildrop and which persists across sessions. The server should never reuse a unique-id in a given maildrop, for as long as the entity using the unique-id exists.

Note that messages marked as deleted are not listed.

### Data Type

#### Integer

#### See Also

Action Property.

## **Urgent Property**

### **Applies To**

TCP, Telnet.

### **Description**

The Urgent Property, when set to True, specifies that all data sent using the Send Property will be sent as TCP Urgent or Out-Of-Band data. Write-only at run time.

### **Usage**

[*form.*][*control.*]**Urgent = True**

### **Remarks**

This property generally should not be used except in special cases.

### **Data Type**

**Integer (Boolean)**

## **User Property (FTP)**

### **Applies To**

FTP.

### **Description**

Specifies the user name for an account, used when logging in.

### **Usage**

*[form.][control.]User = stringexpression*

### **Remarks**

This must be set before using the LoginHost property.

### **Data Type**

**String**

### **See Also**

Account Property; Password Property.

## **User Property (POP3)**

### **Applies To**

POP3.

### **Description**

Specifies the user name to be used for authentication. Must be set prior to setting Action=CONNECTCOMM.

### **Usage**

[*form.*][*control.*]**User** [ = *stringexpression* ]

### **Remarks**

USER/PASS authentication is used unless the Secret property is set, indicating that APOP authentication is to be used first.

### **Data Type**

**String**

## **Verify Property**

### **Applies To**

SMTP.

### **Description**

Asks the mail receiver to confirm that the argument identifies a mail address. If it is a user name, the full name of the user (if known) and the fully specified mailbox are returned.

### **Usage**

*[form.][control.]Verify = stringexpression*

### **Remarks**

This command asks the receiver to confirm that the argument identifies a user. If it is a user name, the full name of the user (if known) and the fully specified mailbox are returned. This command has no effect on the reverse-path buffer, the forward-path buffer, or the mail data buffer.

### **Data Type**

**String**

## **WillOption, WontOption Properties**

### **Applies To**

Telnet.

### **Description**

The WillOption and WontOption properties, when set, send a command to the other end of the connection that you will or won't support an option. These properties are not displayed at design time and write-only at run time.

### **Usage**

[*form.*][*control.*]WillOption = *numericexpression*

[*form.*][*control.*]WontOption = *numericexpression*

### **Remarks**

When you set the WillOption or WontOption properties, a command is immediately sent across the connection to command the other end to do the option. This tells the other end of the connection, "I will/won't support option x." The other end may respond with either a Do or a Don't command, agreeing or disagreeing. This command will generate a Cmd event, with the command they sent as an argument.

The different options you may send are defined in the POWERTCP.BAS constants file.

Note that these properties should not be used if AutoOption is set to False.

### **Data Type**

Integer

### **See Also**

Cmd Event; DoOption and DontOption Properties; DoSubOption Property; SubOption Property.



## **Accept Event**

### **Applies To**

TCP, Telnet.

### **Description**

The Accept event is generated when a remote computer sends a connect message to a listening control.

### **Syntax**

**Sub** *ctlname*\_Accept ( *NewSession* **As Long** )

### **Remarks**

This event will be generated when the control is in listening mode and a remote computer is trying to connect to it. See the Session property for more information and details on how to respond to it.

### **See Also**

Session Property; Listen Event; Action Property (TCP, Telnet).

## **Attach Event**

### **Applies To**

SMTP.

### **Description**

The Accept event is generated periodically during the transfer of a mail message and its attachments.

### **Syntax**

**Sub** *ctlname*\_Attach ( *FileSpec* **As String**, *PercentComplete* **As Long** )

### **Remarks**

This event will be generated when the control is transferring a mail message and attachments to a remote host. The PercentComplete is for the ENTIRE mail message. The FileSpec will change as the attachments are sent. PercentComplete of 100 indicates that the entire message and all attachments have been sent.

### **See Also**

Action Property (SMTP).

## **Click Event**

### **Applies To**

VT.

### **Description**

The Click event is generated when the user clicks on the VT display window.

### **Syntax**

**Sub** *ctlname\_Click* ( *xPos* **As Long**, *yPos* **As Long** )

*xPos* x position of pointer in client coordinates when user clicked left button

*yPos* y position of pointer in client coordinates when user clicked left button

## Cmd Event

### Applies To

Telnet.

### Description

The Cmd event is generated when a control sends you a Telnet option negotiation code.

### Syntax

**Sub** *ctlname\_Cmd* ( *Cmd As Integer*, *TelnetOption As Integer*, *SubOption As String* )

### Remarks

When the Cmd event is generated, you should respond with the DontOption, DoOption, DoSubOption, WillOption, and WontOption properties.

The following table describes the different commands you may receive in the Cmd argument:

Setting	Description
---------	-------------

GO_AHEAD_CMD	The other end of the connection has received the data, and is telling you that you may send more. You can eliminate the necessity of this command by negotiating the SUPPRESS_GO_AHEAD option.
--------------	--

WILL_CMD, WONT_CMD	These two commands tell you that the sender will or won't support a certain option. You can agree or refuse by sending the DoOptionDoOption or DontOptionDoOption command.
--------------------	--

DO_CMD, DONT_CMD	These two commands tell you to enable or disable a certain option. You may enable or disable the option yourself, or you can refuse by sending the WillOptionWillOption or WontOptionWillOption command.
------------------	--

SB_CMD	This command tells you that the sender will negotiate a sub-option.
--------	---

The POWERTCP.BAS constants file defines the different Telnet option codes you may receive.

### Example

See the Tutorial for an example of skeleton option negotiation code.

### See Also

Cmd Property; DontOption Property; DoOption Property; DoSubOption Property; WillOption Property; WontOption Property.

## **Connect Event (TCP, Telnet, SMTP)**

### **Applies To**

TCP, Telnet, SMTP.

### **Description**

The Connect event is generated when a control successfully connects to a remote computer, or when you accept a remote connection from another computer.

### **Syntax**

**Sub** *ctlname*\_Connect ( )

### **Remarks**

The Connect event lets you execute code immediately after you connect to a remote computer, or after a remote computer connects to you. The RemoteHost, RemotePort, LocalDotAddr, and LocalPort properties will be updated with current information and may be read.

### **See Also**

Session Property; Action Property (TCP, Telnet), Action Property (SMTP), Connect Event (FTP).

## Connect Event (FTP)

### Applies To

FTP.

### Description

The Connect Event is generated when the user's control connection establishes contact with the FTP server.

### Syntax

**Sub** *ctlname*\_Connect ( *RemoteDotAddr* As **String**, *RemotePort* As **Long**, *LocalDotAddr* As **String**, *LocalPort* As **Long**, *LocalName* As **String** )

*RemoteDotAddr*      Address of FTP server in xxx.xxx.xxx.xxx notation.

*RemotePort*      Remote port assigned to connection.

*LocalDotAddr*      Local address in xxx.xxx.xxx.xxx notation.

*LocalPort*      Local port assigned to connection.

*LocalName*      The name of the local host computer.

### Remarks

All parameters passed in the event can only be found in this event. Therefore, if an application needs to access this information at a later point, it must be saved in a global variable.

### See Also

Reply Event

## **Connect Event (POP3)**

### **Applies To**

POP3.

### **Description**

The Connect event is generated when a control successfully connects to a remote computer, or when you accept a remote connection from another computer.

### **Syntax**

**Sub** *ctlname*\_Connect ( )

### **Remarks**

The RemoteHost and RemotePort properties will be updated with current information and may be read. The POP3 control will take care of sending authentication messages (USER, PASS, APOP), so you may only want to update status information from within this event.

### **See Also**

Action Property

## Connect Event (TFTP)

### Applies To

TFTP.

### Description

The Connect event is generated when a TFTP client or server is successfully established.

### Syntax

**Sub** *ctlname*\_Connect ( *LocalPort* **As Long**, *LocalName* **As String**, *MaxByteCnt* **As Long** )

*LocalPort*      Local port number assigned to the session.

*LocalName*     Domain name of the network interface used.

*MaxByteCnt*    Maximum number of bytes supported by the UDP transport for a single  
Datagram.

### Remarks

The Connect event lets you execute code immediately after you establish a TFTP client. The LocalDotAddr property will be updated with current information and may be read.

### See Also

Action Property



## Connect Event (UDP, SNMP)

### Applies To

UDP, SNMP.

### Description

The Connect event is generated when a UDP session is successfully established.

### Syntax

**Sub** *ctlname\_Connect* ( *LocalName* **As String**, *MaxByteCnt* **As Long** )

*LocalName* Domain name of the network interface used.

*MaxByteCnt* Maximum number of bytes supported by the UDP transport for a single Datagram.

### Remarks

The Connect event lets you execute code immediately after you establish a UDP session. The LocalDotAddr and LocalPort properties will be updated with current information and may be read.

### See Also

Action Property (SNMP); Action Property (UDP)

## Exception Event

### Applies To

TCP, Telnet, FTP, SMTP, POP3, UDP, SNMP, TFTP.

### Description

The Exception event is generated when an error occurs.

### Syntax

**Sub** *ctlname*\_Exception ( *ErrorCode* **As Integer**, *ErrorDesc* **As String** )

### Remarks

The POWERTCP.BAS constants file defines the different error codes you can receive. The table below lists all possible values.

<b>ErrorCode</b>	<b>Description</b>
PT_OK = 0	OK - check for advisory text message.
PT_HARDWARE = 1	Hardware failure.
PT_PROTOCOL = 2	Protocol software failure.
PT_BADNAME = 3	Name of host cannot be resolved to address.
PT_CONNREFUSED = 4	Connection to host refused - remote host is not listening.
PT_NOROUTE = 5	No route to host (check network part of address).
PT_NOHOST = 6	Remote host is not available (check host part of address).
PT_NOMEM = 7	Insufficient local resources to allocate resources, accept buffer or create channel.
PT_ADDRINUSE = 8	Address/port in use. Could indicate a corrupted transport layer.
PT_NOTCONNECTED = 9	Attempt made to use a session that is not in the PT_CONNECTED state.
PT_NORESOURCE = 10	Insufficient resources on remote host. TFTP error code 3.
PT_NOTACCEPTED = 11	Application did not properly accept passive connection.
PT_SOFTWARE = 12	General software error.
PT_REMOTECLOSE = 13	Remote host reset or closed connection. A Receive Event will follow, indicating a closed connection.
PT_WARNING = 14	General warning. Not currently used.
PT_ERROR = 15	General error. TFTP error code 0.
PT_NOFILE = 16	TFTP error code 1.
PT_ACCESSVIO = 17	TFTP error code 2.
PT_ILLEGALOP = 18	TFTP error code 4.
PT_UNKNOWNID = 19	TFTP error code 5.
PT_FILEEXISTS = 20	TFTP error code 6.
PT_NOUSER = 21	TFTP error code 7.
PT_INUSE = 22	Attempt made to open a session that is already in use.
PT_NOWINSOCK = 23	Error encountered accessing WINSOCK.DLL or WSOCK32.DLL.

### See Also

Action Property (TCP, Telnet); Action Property (SMTP); Action Property (POP3); Action Property (UDP); Action Property (SNMP); Action Property (TFTP).

## File Event

### Applies To

POP3.

### Description

The File event is generated when a file is closed that contains a mail message or a decoded file attachment.

### Syntax

**Sub** *ctlname\_File* ( *FileSpec* **As String**, *Mode* **As String**, *FileSize* **As Long** )

### Remarks

PowerTCP calls this function to notify the application of the successful receipt of a mail message and each attached file.

PowerTCP calls this function to notify the application of the successful spooling of the mail message or attached file. The received file is closed before this event is called. If mail message spooling is not enabled then no File Event is generated related to the message body, since no file is created. Note that mail message data is always returned by Pop3 Event.

The *EncodeType* is for informational purposes only. All data is fully decoded prior to spooling or sending to the user.

If the size of the file created exceeds the maximum count represented by an Unsigned Long, the *FileSize* parameter returned will not be valid.

### See Also

MailFileSpec Property; AttachmentDir Property; Pop3 Event.

## OnHeader Event

PowerTCP calls this function after receiving and parsing the header portion of a mail message.

### Applies To

[POP3](#)

### Syntax

*Object* **OnHeader** ( *nLastCommand*, *sReturnPath*, *sReceived*, *sDate*, *sFrom*, *sSubject*, *sSender*, *sTo*, *sCc* )

---

- nLastCommand* An **Integer** value that specifies the last command sent to a POP3 server - either POP3\_RETR or POP3\_TOP, refer to the [CommandConstants](#) for details.
- sReturnPath* A **String** containing the mail header ReturnPath field or zero length string.
- sReceived* A **String** containing the mail header Received field or zero length string.
- sDate* A **String** containing the mail header Date field or zero length string.
- sFrom* A **String** containing the mail header From field or zero length string.
- sSubject* A **String** containing the mail header Subject field or zero length string.
- sSender* A **String** containing the mail header Sender field or zero length string.
- sTo* A **String** containing the mail header To field or zero length string.
- sCc* A **String** containing the mail header Cc field or zero length string.

### Remarks

PowerTCP calls this function to notify the application of the receipt and parsing of a mail message header. Header Event returns eight common fields as strings, whereas Pop3 Event (when status = POP3\_HEADER) returns the complete header for you to parse. Either or both events can be used. Calls to Header Event will be generated in response to successful **Retrieve** and **Top** property settings.

The fields returned by Header Event may be folded - containing embedded CRLF<whitespace>. Further, if a mail message contains duplicate header fields, the sequence of duplicates (including fields field name) will be returned as a single string. If a field has not been located, a zero length string is returned.

**Warning:** if Header Event is to be utilized, you must not modify the buffer returned by Pop3 Event (when status=POP3\_HEADER), as this will invalidate the parsing of the header fields returned by . Header Event.

## HostCommand Event

### Applies To

VT.

### Description

The HostCommand event is generated when the host commands the terminal to perform an action. Unless Ignore is set to TRUE, the VT control will effect the commanded change.

### Syntax

**Sub** *ctlname*\_HostCommand ( *Attribute As Integer, Value As Long, Ignore As Integer* )

*Attribute* Attribute being set or reset.

*Value* Set (TRUE) or reset (FALSE).

*Ignore* Set to TRUE to defeat the instruction from the host.

### Remarks

Most applications will not require any code for this event.

The possible values for *Attribute* are defined in the POWERTCP.BAS file, with descriptions.

## KeyDown Event

### Applies To

VT.

### Description

The KeyDown event is generated when the user presses a key.

### Syntax

**Sub** *ctlName*\_KeyDown ( *KeyCode* **As Integer**, *Shift* **As Integer**, *Extended* **As Integer** )

### Remarks

This event provides a preview of the key so that the program can change the virtual key code or shift attribute to a different value by setting *KeyCode* to a new value (for key mapping applications). You can also set *KeyCode* to 0 (negating any processing by the control) and send your own substitution string to the host. If *KeyCode* is not equal to zero when the event finishes, the control will send a key sequence for the current value of *KeyCode*.

*KeyCode* is any valid virtual key code. *Shift* is a bitwise OR of :

- 1 - Shift key depressed
- 2 - Control key depressed

*Extended* is 1 if the key is part of the extended keyboard. For example, both Enter keys have the same virtual key code, but the Enter key on the numeric keypad would have a value of 1 for the *Extended* parameter.

### See Also

KeyPress Property.

## **KeyPress Event**

### **Applies To**

VT.

### **Description**

The KeyPress event is generated when the emulator has data to be sent to the host.

### **Syntax**

**Sub** *ctlname*\_KeyPress ( *KeyString* **As String** )

### **Remarks**

This event provides the actual data that should be sent to the host program. You should use the KeyDown Event for any key substitution applications.

### **See Also**

KeyDown Property.

## **Listen Event**

### **Applies To**

TCP, Telnet.

### **Description**

The Listen event is generated when the control enters listening mode, by setting the Action property to Listen (Action = LISTENCOMM).

### **Syntax**

**Sub** *ctlname*\_Listen ( )

### **Remarks**

The Listen event is a confirmation that the control entered listening mode. When the Action property is set to Listen, the Listen event will be generated if the control was successful in entering Listen mode.

### **See Also**

Accept Event, Session Property, Action Property.



## **Log Event (FTP)**

### **Applies To**

FTP.

### **Description**

The Log Event is used to notify the application of FTP commands sent by PowerTCP and status information that are not part of the FTP protocol. These descriptive messages may be posted into an edit box for review during the development process.

### **Syntax**

**Sub** *ctlname\_Log* ( *Message* As **String** )

### **Remarks**

Log messages are advisory only.

## **Log Event (SMTP, POP3)**

### **Applies To**

SMTP, POP3.

### **Description**

The Log event is generated when the PowerTCP library sends a message to the server.

### **Syntax**

**Sub** *ctlname\_Log* ( *Cmd As String* )

### **Remarks**

Command is the exact command sent to the server. It is useful for displaying activity generated by the PowerTCP library.

## **NewLine Event**

### **Applies To**

VT.

### **Description**

The NewLine event occurs when the cursor changes rows. This property is useful for capturing data to a disk file or other destination.

### **Syntax**

**Sub** *ctlname*\_NewLine ( *LineText* As **String** )

### **Remarks**

If you are implementing logging (capturing), you should write LineText to the disk file. LineText contains the line of text in which the cursor was positioned before it moved. Note that LineText ends with a CR/LF (Chr\$(13) & Chr\$(10)), so that if the Print # statement is used to send it to a file, the statement should end with a semicolon.

### **Example**

```
Sub VT1_NewLine (LineText As String)
    ' This will send the text to file #1, which must have
    ' been previously opened by the program.
    Print #1, LineText;
End Sub
```

## Pop3 Event

Applies To

POP3.

### Description

PowerTCP calls **Pop3 Event** when a reply has arrived from the POP3 server. The body of this event is where you must insert all your control code to interpret the success and/or failure of the commands sent to the POP3 server.

### Syntax

**Sub** *ctlname\_Pop3* ( *Status As Integer, LastCommand As Integer, ReplyStr As String, FileSpec As String, Mode As String, PercentComplete As Integer* )

*Status* Status or State of POP3 dialog with host. See table below.

*LastCommand* Last command sent to POP3 server. See descriptions below.

*ReplyStr* Data sent from POP3 server responding to last command sent. PowerTCP may call this method numerous times while retrieving a single mail message.

*FileSpec* Specification of file currently being transferred. If spooling the mail header/body to a file, *Status* would be equal to POP3\_MAIL. If spooling an attachment to a file, *Status* would be equal to POP3\_FILE.

*Mode* The file encoding method if *Status* = POP3\_FILE. The codes are the following:

M - MIME Base 64 Encoding.

U - UU Encoding.

N - No Encoding.

*PercentComplete* A number between 0 and 100 specifying the percent retrieved of the complete Message(i.e. Mail plus all attachments). When Retrieving a mail message, the only indication that the message is complete is *PercentComplete* = 100 and *Status* = POP3\_FILE or POP3\_MAIL.

### Remarks

You will initiate a POP3 session by calling the Connect method, after which all replies from the POP3 server will cause PowerTCP to call Pop3 event. Besides providing a buffer that contains the raw data sent from the POP3 server, PowerTCP handles interpretation of the stream, calling this method with parameters that provide useful information for the developer. PowerTCP calls the Pop3 event when a reply has arrived from the POP3 server in response to a POP3 message being sent to the server, or the connection is closed. The body of this function is where you place all your control and status code to interpret the success and/or failure of the commands sent to the POP3 server.

The *LastCommand* parameter is used to determine what method was used last, so you know how to interpret the reply. Then check *Status* to find out if the reply is a standard POP3 reply (pop3\_ReplyPos or pop3\_ReplyNeg), the header of a mail message (pop3\_Header), part of a mail message body (pop3\_Mail), or part of a file attachment (pop3\_File).

When *Status* = pop3\_Header in response to a Retrieve request, the *ReplyStr* contains the entire mail header. Note that most header fields are more easily available from the Header event which returns common fields as separate null terminated strings. Generally, the Pop3 Event with *Status* = pop3\_Header can be ignored, and the header information retrieved from the Header Event call which will immediately follow Pop3 Event. Note that the *ReplyStr* buffer returned from Pop3 Event must not

be modified if it is intended to use the Header Event

In response to TopLines , an Pop3 Event) with Status = pop3\_Header is not generated. This response will be pop3Mail since it can contain header and message data if more than 0 lines were requested from the file. In this case, Header Event will still provide the correct parsed header data.

When Status =pop3\_File, decoded data of an attached file is pointed to by *ReplyStr*. This response occurs only after Retrieve request when attachments are included with the mail. Repeated calls to Pop3 Event will be made with this status until all of the attachment file is processed. If spooling is enabled *FileSpec* contains the name of the file storing the data, and the data buffer has already been spooled to disk prior to this call. The *FileSpec* used for spooling attachments may not match the filename described in the mail message headers if the NoOverWrite *Property* is set. This is discussed in the Connect() method description.

Note that *PercentComplete* cannot be used to detect completion of a particular attached file, although the completion of all attachments is signaled by *PercentComplete* equal 100. The completion a single attachment file is signaled by File Event.

The Mail Message is spooled to disk if the *MailFileSpec* parameter of the Retrieve method is not NULL. File attachments are automatically decoded and spooled to disk if the *AttachmentDir* parameter of the Retrieve function is not NULL. Attachment files are complete and closed when PowerTCP calls the File Event.

You can monitor the progress of the entire transfer as the Status value goes from pop3\_ReplyPos (the initial indication that mail will follow), to pop3\_Header, to pop3\_Mail (possibly called several times while the mail message is spooled), to pop3\_File (probably called multiple times while *FileSpec* changes to indicate the receipt of several files). While Status is pop3\_Mail or pop3\_File, *PercentComplete* will be increasing from 0 or more up to 100. A value of 100 is positive indication that the mail and all attachments have been received, and another POP3 command can be issued.

The mail message is spooled to disk if the *MailFileSpec* parameter of the Retrieve method is not NULL. File attachments are automatically decoded and spooled to disk if the *AttachmentDir* parameter of the Retrieve method is not NULL (they are also reported when PowerTCP calls the File event).

A closed connection is confirmed when *nStatus* = pop3\_Closed and *nReply* =NULL.

### **POP3 Status Description**

POP3\_CLOSED = 0 - The connection to the POP3 Server is closed.

POP3\_REPLY\_POS = 1 -. This status is generated as the first response to any successful command sent to the POP3 server. (The POP3 server has responded with a "+OK"). *Reply* provides data associated with the response, which may be multi-line, as indicated by *PercentComplete* not equal 100. Such a multi-line response will ultimately be terminated by a Pop3 Event call with *PercentComplete* equal 100.

POP3\_REPLY\_NEG = 2 - The POP3 server has responded with "-ERR", indicating a Negative reply to the previous message sent to the server.

POP3\_MAIL = 3 - Data of a mail message is pointed to by. Occurs in response to LIST, TOP or RETR commands. Repeated calls to Pop3 Event will be made with this status until all of the mail body is processed. When this status occurs with *PercentComplete*=100, that signals completion of the mail message (e.g. a CRLF, ".",CRLF sequence has been received).

POP3\_FILE = 4 - When a file is being written, the *Filename* parameter contains the current

file name. The file data is specified by the *FileSpec* parameter.

POP3\_HEADER = 5 - A complete mail header has been assembled and is pointed to by *ReplyStr*. Occurs in response to RETR commands only. One header is returned per call. See Remarks.

<b>POP3 Commands</b>	<b>Description</b>
----------------------	--------------------

POP3_APOP = 0	Sent during authentication process by PowerTCP.
POP3_DELE = 1	Sent by the <u>DeleteMsg</u> property.
POP3_LIST = 2	Sent by the <u>List</u> property.
POP3_NOOP = 3	Sent by the <u>Action</u> property.
POP3_PASS = 4	Sent by the <u>Action</u> property.
POP3_QUIT = 5	Sent by the <u>Action</u> property.
POP3_RSET = 6	Sent by the <u>Action</u> property.
POP3_RETR = 7	Sent by the <u>Retrieve</u> property.
POP3_STAT = 8	Sent by the <u>Action</u> property.
POP3_TOP = 9	Sent by the <u>TopLines</u> property.
POP3_UIDL = 10	Sent by the <u>Uidl</u> property.
POP3_USER = 11	Sent by the <u>Action</u> property.
POP3_CONNECT = 12	Used after a TCP connection is established with a POP3 server to indicate the initial greeting from the server.

**See Also**

File Event.

## Recv Event (TCP, Telnet, FTP)

### Applies To

[TCP](#), [Telnet](#), [FTP](#).

### Description

The Recv event is generated when the other end of a connection sends data. It can also signal that the connection has been closed.

### Syntax

**Sub** *ctlname\_Recv* ( *RecvData* **As String** )

### Remarks

When one end of a connection uses the [Send](#) property to send a string of text, the other end will generate at least one Recv event. The RecvData argument will contain the text sent.

If the control does not respond to the Recv event, the received string will be lost. Therefore, the program should either respond to the string, or store it in a global variable for later use.

The data contained within the RecvData parameter **may** include null characters within it.

### Important

If the length of RecvData is 0, this signals that the connection has been closed.

### FTP Specific

Data is received in this event in response to the [List](#), [NameList](#), and [Retrieve](#) properties. The receipt of this event with the length of RecvData equal to 0 signals the closing of the data connection, implying that the file should be closed or other termination action be accomplished.

Note that when retrieving a file, the Recv event is called only if the [LocalFileSpec](#) property is set to an empty string ("").

### Example

The following example places received data into a text box and notifies the user when the connection goes down:

```
Sub TCP1_Recv (RecvData As String)
    If RecvData = "" Then
        ' The connection has terminated
        MsgBox "The connection has been closed."
    Else
        Text1.Text = Text1.Text & RecvData
    End If
End Sub
```

### See Also

[Send](#) Event, [Send](#) Property.

## **Recv Event (UDP)**

### **Applies To**

UDP

### **Description**

The Recv event is generated when a Datagram is received or the socket is closed.

### **Syntax**

**Sub** *ctlname\_Recv* ( *RecvData* **As String**, *RemoteDotAddr* **As String**, *RemotePort* **As Long** )

### **Remarks**

When a Datagram is received PowerTCP will generate the Recv event. The RecvData argument contains the text received. The RemoteDotAddr string contains the dot address of the source host, and RemotePort contains the source host port. Your program should either use the string, or store it in a global variable for later use.

The data contained within the Data parameter may include null characters within it.

If the length of RecvData is 0 (RecvData = ""), this signifies that the State is now PT\_CLOSED.

### **See Also**

Send Event, Send Property.



## RecvSnmp Event

### Applies To

SNMP.

### Description

The RecvSnmp event is generated when an SNMP agent sends a response to your query, or when your SNMP agent receives a query from an SNMP manager.

### Syntax

**Sub** *ctlname\_RecvSnmp* ( *MessageType As Integer* )

*MessageType* Type of message received.

### Remarks

When a query is received by your agent or a response to a query is received by your manager, PowerTCP will generate the RecvSnmp event. Possible values for *MessageType* are included in POWERTCP.BAS:

SNMP\_GET\_REQUEST (4) - Your agent applications has received a request or information.

SNMP\_GET\_NEXT\_REQUEST (5) - Your agent application has received another request.

SNMP\_SET\_REQUEST (6) - Your agent has received data to set a local value.

SNMP\_GET\_RESPONSE (7) - Your manager has received a response from a remote manager.

The RemoteHost string contains the dot address of the SNMP agent. Information from the agent is stored in the following properties just before this event is fired: Community, ErrorStatus, ErrorIndex, RemoteHost (in dot address notation), RemotePort, RequestID, nObjects, ObjectID, ObjectValue and ObjectType. All of these values must be copied elsewhere if they are to be kept as they will be overwritten in the next RecvSnmp or RecvTrap Event.

### See Also

Send Event, Action Property.

## **RecvTrap Event**

### **Applies To**

SNMP.

### **Description**

The Trap event is generated when the SNMP agent sends you an unsolicited trap message.

### **Syntax**

**Sub** *ctlname\_RecvTrap* ()

### **Remarks**

This event is fired whenever a trap packet arrives on the LocalPort.

When a trap is received from an SNMP agent, PowerTCP will generate the Trap event. The RemoteHost Property string contains the address of the SNMP agent. Additional information from the agent is stored in the following properties just before this event is fired:

Community, Enterprise, NodeDotAddr, GeneralTrap, SpecialTrap, TimeStamp, nObjects, RemoteHost (in dot address format), RemotePort, ObjectID, ObjectValue, and ObjectType.

## Reply Event

### Applies To

FTP.

### Description

PowerTCP calls the Reply Event when a reply has arrived on the control connection. The body of this event is where you must insert all your control code to interpret the success and/or failure of the commands sent to the FTP server.

### Syntax

**Sub** *ctlname\_Reply* ( *Status As Integer*, *LastCommand As Integer*, *ReplyCode As Integer*, *ReplyStr As String* )

*Status* Status of FTP dialog.

*LastCommand* Last command sent to FTP server (enumerated type).

*ReplyCode* FTP protocol return code (integer value of first 3 digits of ReplyStr).

*ReplyStr* NULL-terminated reply string

### Remarks

The Status parameter indicates how PowerTCP has interpreted the result. A table describing these values follows below.

The LastCommand parameter specifies the last command sent out over the connection (assuming a PowerTCP property was used, and not the Command property). This means that you do not have to save the command which was sent last.

The ReplyCode parameter is a value between 110 and 553 that indicates the status of the FTP server. The meaning of these values can be found in RFC 959 (included in the PowerTCP SDK software distribution). This value is provided for your convenience if detailed debugging is required.

The ReplyStr parameter is the raw reply string passed back from the FTP server. We make this easier because we buffer up a complete reply and NULL terminate it to make it easier to interpret. ReplyStr can be a multi-line string separated by CR/LF pairs. No CR/LF is included on the last line.

You will put most of your code within this event. Typically, you should check for the appropriate LastCommand and an FTP\_SUCCESS status as indicators. For example, if retrieving multiple files, check for LastCommand = FTP\_RETR and Status = FTP\_SUCCESS to determine when to send the next one. Please refer to the included FTP sample applications to see how this is easily implemented.

The following table describes the possible status indications:

#### **FTP\_STATUS Description**

---

FTP\_UNKNOWN = 0 Command Property was used for sending last command (in lieu of another PowerTCP property), or the FTP server has volunteered spontaneous information (such as a system shutdown message)

FTP\_SUCCESS = 1 The sent FTP command (or sequence of commands sent by PowerTCP) has completed successfully.

FTP\_ERROR = 2 The sent FTP command has generated an error condition. This message indicates bad synchronization of commands over the control connection.

FTP\_FAILURE = 3 The sent FTP command has generated a failure condition. This message can be expected due to a security access error or other failure encountered

under normal conditions.

FTP\_WORKING = 4 The send FTP command has completed successfully, but PowerTCP is sending a follow-on command to complete a sequence of command. Sequences are generated by properties like LoginHost, Rename, Retrieve and Store.

The FTP\_COMMAND enumerated types follow:

<b>FTP_COMMAND</b>	<b>Description</b>
FTP_CLOSED = 0	Control connection is closed after <u>LogOut</u> property is used.
FTP_CLEAR = 1	Command property used or reply has cleared last command.
FTP_USER = 2	See <u>User</u> property.
FTP_PASS = 3	See <u>Password</u> property.
FTP_ACCT = 4	See <u>Account</u> property.
FTP_CWD = 5	See <u>ChDir</u> property.
FTP_CDUP = 6	See <u>ChDirUp</u> property.
FTP_SMNT = 7	See <u>StructMount</u> property.
FTP_QUIT = 8	See <u>LogOut</u> property.
FTP_REIN = 9	See <u>Reinitialize</u> property.
FTP_PORT = 10	See <u>Port</u> property.
FTP_PASV = 11	See <u>Passive</u> property.
FTP_TYPE = 12	See <u>Type</u> property.
FTP_STRU = 13	See <u>StructMount</u> property.
FTP_MODE = 14	See <u>Mode</u> property.
FTP_RETR = 15	See <u>Retrieve</u> property.
FTP_STOR = 16	See <u>Store</u> property.
FTP_STOU = 17	See <u>StoreUnique</u> property.
FTP_APPE = 18	See <u>Appe</u> property.
FTP_ALLO = 19	See <u>Allocate</u> property.
FTP_REST = 20	See <u>Restart</u> property.
FTP_RNFR = 21	See <u>Rename</u> property.
FTP_RNTO = 22	See <u>Rename</u> property.
FTP_ABOR = 23	See <u>Abort</u> property.
FTP_DELE = 24	See <u>Dele</u> property.
FTP_RMD = 25	See <u>RemoveDir</u> property.
FTP_MKD = 26	See <u>MakeDir</u> property.
FTP_PWD = 27	See <u>PrintWorkingDir</u> property.
FTP_LIST = 28	See <u>List</u> property.
FTP_NLST = 29	See <u>NameList</u> property.
FTP_SITE = 30	See <u>Site</u> property.
FTP_SYST = 31	See <u>System</u> property.
FTP_STAT = 32	See <u>Status</u> property.
FTP_HELP = 33	See <u>Help</u> property.
FTP_NOOP = 34	See <u>Noop</u> property.

**See Also**

Send Event, Send Property.

## **Send Event (TCP, Telnet, FTP, UDP)**

### **Applies To**

TCP, Telnet, FTP, UDP.

### **Description**

The Send event is generated when the PowerTCP control successfully buffers a string to the system buffers.

### **Syntax**

**Sub** *ctlname\_Send* ( *DataTag* **As Long** )

*DataTag*        The value of the DataTag property when the data was sent.

### **Remarks**

This event is generated when the system network buffers have successfully accepted data for transmission.

### **FTP Specific**

When the Store, StoreUnique, or Appe properties are used, the PowerTCP library and FTP server accomplish the steps to establish a data connection. When established, PowerTCP calls the Send event to signal that the application may now start sending data by using the Send property. Note that this event is generated only if the LocalFileSpec property is set to an empty string (""). DataTag is a value that indicates how many bytes were sent.

### **See Also**

Recv Event; Send Property; DataTag Property.

## Send Event (SNMP)

### Applies To

SNMP.

### Description

The Send event is generated when the PowerTCP control successfully submitted a string to the system buffers.

### Syntax

**Sub** *ctlname\_Send* ( *RequestID* **As Long** )

*RequestID* Reflects the value of the RequestID Property when the Action Property was used to send an SNMP packet.

### Remarks

PowerTCP uses RequestID to tag all SNMP packets sent using the Action property: SNMP\_GET\_REQUEST, SNMP\_GET\_NEXT\_REQUEST, SNMP\_SET\_REQUEST, SNMP\_GET\_RESPONSE, or SNMP\_TRAP\_MESSAGE.

### See Also

Action Property.

## Smtp Event

### Applies To

SMTP.

### Description

PowerTCP calls the Smtp Event when a reply has arrived from the mail receiver. The body of this event is where you must insert all your control code to interpret the success and/or failure of the commands sent to the mail receiver.

### Syntax

**Sub** *ctlname\_Smtp* ( *Status* **As Integer**, *LastCommand* **As Integer**, *ReplyCode* **As Integer**, *ReplyStr* **As String**, *Complete* **As Integer** )

*Status* Status of SMTP dialog with host.

*LastCommand* Last command sent to SMTP server (enumerated type).

*ReplyCode* SMTP protocol return code (integer value of first 3 digits of ReplyStr).

*ReplyStr* NULL-terminated reply string

*Complete* True if last line of multi-line reply.

### Remarks

PowerTCP calls the Smtp event when a reply has arrived on the SMTP connection or the connection is closed. The body of this event is where you must insert all your control code to interpret the success and/or failure of the commands sent to the SMTP server.

The Status parameter indicates how PowerTCP has interpreted the result. A table describing these values follows below.

The LastCommand parameter specifies the last command sent to the SMTP server.

The ReplyCode parameter is a value between 211 and 554 that indicates the status of the SMTP server. The meaning of these values can be found in RFC 821 (included in the PowerTCP SDK software distribution). These values need only be interpreted if protocol problems are encountered.

The ReplyStr parameter is the raw reply string passed back from the SMTP recipients. PowerTCP makes this easier because it buffers up a complete line and NULL terminates it to make it easier to interpret. ReplyStr can be a multi-line string separated by CR/LF pairs. No CR/LF is included on the last line.

A closed connection is confirmed when Status = SMTP\_CLOSED.

The following tables describes the possible status and command indications:

#### **SMTP StatusDescription**

---

SMTP\_CLOSED = 0 The connection is closed.

SMTP\_SUCCESS = 1 The sent SMTP command (or sequence of commands sent by PowerTCP) has completed successfully.

SMTP\_ERROR = 2 The sent SMTP command has generated an error condition. This message indicates bad synchronization of commands over the channel.

SMTP\_FAILURE = 3 The sent SMTP command has generated a failure condition. This message can be expected due to a security access error or other failure encountered under normal conditions.

SMTP\_WORKING = 4 The SMTP command has completed successfully, but PowerTCP is sending a follow-on command to complete a sequence of commands. Sequences are



generated by calling methods like Mail.

<b>SMTP Commands</b>	<b>Description</b>
SMTP_DATA = 0	Sent when <u>Action</u> = SEND_MAIL is set.
SMTP_EXPN = 1	Sent when <u>Expand</u> property is set.
SMTP_HELO = 2	Sent when <u>Action</u> = CONNECTCOMM is set.
SMTP_HELP = 3	Sent when <u>Help</u> property is set.
SMTP_MAIL = 4	Sent when <u>Action</u> = SEND_MAIL is set.
SMTP_NOOP = 5	Sent when <u>Action</u> = SEND_NOOP_MAIL is set.
SMTP_QUIT = 6	Sent when <u>Action</u> = CLOSECOMM is set.
SMTP_RCPT = 7	Sent when <u>Action</u> = SEND_MAIL is set.
SMTP_RSET = 8	Sent when <u>Action</u> = RESET_MAIL is set.
SMTP_SAML = 9	Sent when <u>Action</u> = SEND_SAML is set.
SMTP_SEND = 10	Sent when <u>Action</u> = SEND_SEND is set.
SMTP_SOML = 11	Sent when <u>Action</u> = SEND_SOML is set.
SMTP_VRFY = 13	Sent when the <u>Verify</u> property is set.
SMTP_CONNECT = 14	First reply from SMTP server after connection is established.

## Tftp Event

### Applies To

TFTP.

### Description

The Tftp event is generated when a Datagram with file data is sent or received.

### Syntax

**Sub** *ctlname\_Tftp* ( *Op As Integer, LocalFileSpec As String, Block As Long, ByteCnt As Integer, TransferID As Long, ActiveCnt As Integer, RemoteDotAddr As String, RemotePort As Long, ErrorCode As Integer, ErrorDesc As String* )

*Op* TFTP\_GET indicates file is being stored locally. TFTP\_PUT indicates file is being read. TFTP\_CLOSED indicates the local socket was just closed.

*LocalFileSpec* LocalFileSpec from file currently used for buffering incoming data.

*Block* Value of 0 signals the start of a file transfer (client and server operation), before any data has been accessed. Value of 1 signals the first packet (512-byte piece) of the file, and increments by one for each subsequent packet. Multiply Block by 512 to determine how much of the file has already been transferred.

*ByteCnt* The byte count of the packet sent. This will be 516 (4 byte header and 512 bytes of data) for all blocks except for the last one. The end of a transfer is known when  $4 \leq \text{ByteCnt} \leq 515$ . The closing of the associated UDP socket is known by a value of 0 (*Op* is also TFTP\_CLOSED).

*TransferID* Counter that identifies transfer taking place. This value starts at 1 and is incremented for each unique file transfer.

*ActiveCnt* Number of active transfers. Will always be 1 for TFTP client operation. Will always be at 0 or more for TFTP server operations.

*RemoteDotAddr* The Internet dot address of the remote host that is involved with the file transfer.

*RemotePort* Port number of remote host that is involved with the file transfer.

*ErrorCode* See the Exception event.

*ErrorDesc* See the Exception event.

### Remarks

This event is called automatically when a TFTP server or client has received or sent file data.

The transfer being reported can be aborted by setting AbortTransfer = *TransferID* from within this event.

When data is received, PowerTCP automatically sends an acknowledgment. If the acknowledgment is not received at the other end, then another data packet will be sent (again because the other side did not receive an acknowledgment). PowerTCP will not write duplicate buffers to disk, but will call **Tftp()** for duplicate packets that are received or sent. This allows you to monitor re-transmits.

A Block value of 0 signals the start of a file transfer (client and server operation), before any data has been accessed. A Block value of 1 is used for the first packet (512-byte piece) of the file, and increments by one for each subsequent packet. Multiply Block by 512 to determine how much of the file has transferred.

### See Also

Send Property.



## Transfer Event

### Applies To

FTP.

### Description

The Transfer event is called when the FTP control automatically sends or receives data during a file transfer.

### Syntax

**Sub** *ctlname\_Transfer* ( *LastCommand* **As Integer**, *BlockCnt* **As Long**, *ByteCnt* **As Long** )

*LastCommand*        The command which initiated the file transfer. This may be FTP\_RETR, FTP\_STOR, FTP\_STOU, or FTP\_APPE.

*BlockCnt*         This specifies which block has been sent. The first block is 1, the second block is 2, and so on. A *BlockCnt* of zero signals that the transfer is complete.

*ByteCnt*         This contains the total number of bytes which have been transferred so far.

### Remarks

This event is only called if the file transfer was handled by the control (signaled by specifying the LocalFileSpec Property). Otherwise, the Send Event and Recv Event are called instead.

### See Also

LocalFileSpec Property; Recv Event; Send Event.

## **Clear Method**

### **Applies To**

VT.

### **Description**

Clears the control of all text.

### **Usage**

[*form.*][*control.*]**Clear**

### **Remarks**

Inserts spaces into the screen display and scrollbar buffer. Positions the cursor at row 1, column 1.

### **Data Type**

**Integer**

### **See Also**

Reset Property

## Additional Resources

RFC stands for Request for Comment. The RFCs are a set of Internetworking documents which define the standards for all Internet protocols. The text of many RFCs has been included with PowerTCP. However, for more information, call 1.800.235.3151 (we're just a little too much into IP addresses here...), or write to:

SRI International, Room EJ291  
333 Ravenswood Avenue  
Menlo Park, CA 94025

In addition, you can send email:

to: rfc-info@ISI.EDU  
subject: getting rfcs

In the body of your mail type:

help: ways\_to\_get\_rfcs

We have also found the following texts to be useful to our programming staff (especially number five):

1. Black, Uyles 1992, *TCP/IP and Related Protocols*, McGraw-Hill, Inc.
2. Comer, Douglas E. & Stevens, David L. 1993, *Internetworking with TCP/IP Volume III - Client-Server Programming and Applications*, Prentice-Hall, Englewood Cliffs, New Jersey
3. Stallings, William 1993, *SNMP,SNMPV2, and CMIP: the practical guide to network management standards*, Addison-Wesley Publishing Company, Reading, Massachusetts
4. Stevens, W. Richard 1990, *UNIX Network Programming*, Prentice Hall, Englewood Cliffs, New Jersey
5. Stevens, W. Richard 1994, *TCP/IP Illustrated, Volume 1 - The Protocols*, Addison-Wesley Publishing Company, Reading, Massachusetts

## Frequently Asked Questions

This section contains information which may help you when trouble-shooting. If you are trying to find a solution to a problem, check in the following places:

- The manual
- Online help (possibly more current than the manual)
- The Read Me file (more current than online help)
- This section (Frequently Asked Questions)
- The sample applications which come with PowerTCP. These contain hundreds of lines of sample code which may contain the answer to your problem.
- Technical support

### **Q. The PowerTCP samples work, but my application doesn't. What should I check for?**

A. The most common problem encountered by Visual Basic PowerTCP programmers is caused by DoEvents. NEVER put a DoEvents statement within a Visual Basic event, as all other event notification will be blocked by Visual Basic (no PowerTCP events will be called). This is a documented "feature" of Visual Basic that is meant to protect against re-entrancy and infinite loops. It has the unfortunate side-effect of disabling all event-driven communication activity.

### **Q. How can I write a TCP server program which accepts many connections at the same time? I don't want to have to create a hundred controls at design time for each accepted connection.**

A. You can create a form with two PowerTCP controls on it. One will handle incoming connections. The other will become a control array for each connection. Its Index property should be set to 0.

When the listening control accepts a connection, it can use the Load statement to create a new PowerTCP control in the array. It can then pass the connection on to this new control.

A variation on this method is to have a listening control on one form, and another control on another form. When the listening control accepts a connection, it can create a new instance of the other form and pass the connection to the control on the newly created form. This is demonstrated in the Echo Server sample which comes with PowerTCP.

### **Q. When I try to connect to a Telnet server using the Telnet custom control, the connection is made but nothing seems to be happening. What could be the problem?**

A. First, make sure that the RemotePort property is set to 23, which is the standard Telnet port. If this does not fix the problem, try running the Telnet sample which comes with PowerTCP. If the sample does not work, then the problem is with the Telnet server or with your TCP stack.

However, if the PowerTCP sample functions properly, and the RemotePort is set to 23, the problem is most likely caused by option negotiation. If the AutoOption property is False, and you have not written any custom option negotiation code, then set AutoOption to true. This will enable automatic option negotiation, and should solve the problem.

If you are using customized option negotiation (AutoOption = False and you respond to the Cmd event), then make sure that you respond to every Will, Won't, Do, Don't, and SubOption message. Many Telnet servers require absolute compliance, and will

not function otherwise. You can use the skeleton code in the tutorial section for option negotiation as a model for your own.

**Q. The VT control is not functioning correctly. When I send data to it, nothing is displayed, and I can't type anything in it.**

A. Make sure that both the Enabled and Cursor properties are set to true.

**Q. Reading from the VT control's Text property is very slow, and I have to read from it repeatedly. What can I do?**

A. When part of the VT's text property is accessed using the Mid\$ function, Visual Basic copies all the VT control's data into a new location and accesses it from there. This can be slow if done repeatedly. If the Text property is assigned to a string variable, and the string variable is accessed, performance can increase by a factor of 2000%, since accessing variables is approximately twenty times faster than accessing properties.

**Q. PowerTCP isn't working at all. The samples programs will load, but they don't send data correctly. In addition, I seem to be getting GPFs every so often. What could be the problem?**

A. First, make sure that you have a Winsock version 1.1-compliant TCP/IP stack on your computer. If you have Windows NT or Windows 95, you can install it using your Windows Setup program. If you have Windows for Workgroups, one is available from Microsoft free of charge. In addition, there are several third-party and shareware stacks available. One widely-used shareware stack is Trumpet Winsock.

If you are certain you have a stack on your computer, make sure you have *only* one. If you have two or more stacks installed (Microsoft and Trumpet, for example), this could be the source of conflicts.

If other TCP/IP applications function correctly, but not PowerTCP, make sure that the PowerTCP files are in the proper directories. For example, if the Visual Basic custom controls are not in the \WINDOWS\SYSTEM directory, the Visual Basic samples will not work.

**Q. Under Delphi and PowerBuilder, some of the data I send becomes corrupted, especially when I use suboption negotiation under Telnet. What could be the problem?**

A. Under Delphi and PowerBuilder, strings are null-terminated. Therefore, if you try to send a string which contains null characters (Chr\$(0)), it will be truncated at that point. The solution is to use the SendString and/or SendByte functions. Setting SendByte to 0 successfully sends a null character.

**Q. I have problems when I try to use the Type property in the FTP control when I use it with PowerBuilder. What might be wrong?**

A. The keyword Type is a reserved word in PowerBuilder, and so a new property has been added to the control - TypeTransfer. This is identical to the Type property, but is PowerBuilder-compatible. The Type property has not been removed from PowerTCP for backwards-compatibility reasons.

**Q. What are all the PTxx.DBG files in my default directory?**

A. If the PT\_DEBUG flag has been set to enable debugging for PowerTCP, all sent and received data is captured into a single file. You may want to delete these files after each time you use debugging with PowerTCP, as some of them may be quite large, depending on the data sent and received.

**Q. When I receive a carriage return in the Recv event and place it in a text box, it shows up as a black square. How can I make it appear as a carriage**



**return?**

A. Carriage returns are sent as Chr\$(13). In Windows, ending a line requires *two* characters--a Chr\$(13) followed by a linefeed character (Chr\$(10)). To solve your problem, you must search through the data you receive for carriage returns and add a linefeed character after each one before you place them in a text box.

**Q. The same thing happens when a backspace character is received—instead of deleting the previous character, it places a black box in the text box I add it to. How can I make it delete the previous character?**

A. Backspace characters are sent as Chr\$(8). You must search for these characters when you receive them, and perform the delete yourself when you find one. For example, if you receive the backspace character, instead of placing it in a text box, you should say `Text1 = Left$(Text1, Len(Text1) - 1)`. This will delete the last character from the text box.

