Serielle Kommunikation mit Visual Basic

Dieses Online-Dokument macht Sie mit den Grundlagen der Datenfernübertragung vertraut und erläutert die serielle Programmierung unter Visual Basic anhand kleiner Programmbeispiele.

Serielle Kommunikation mit Visual Basic

Dieses Online-Dokument macht Sie mit den Grundlagen der Datenfernübertragung vertraut und erläutert die serielle Programmierung unter Visual Basic anhand kleiner Programmbeispiele.

Serielle Kommunikation mit Visual Basic

Dieses Online-Dokument macht Sie mit den Grundlagen der Datenfernübertragung vertraut und erläutert die serielle Programmierung unter Visual Basic anhand kleiner Programmbeispiele.

Serielle Kommunikation mit Visual Basic

Dieses Online-Dokument macht Sie mit den Grundlagen der Datenfernübertragung vertraut und erläutert die serielle Programmierung unter Visual Basic anhand kleiner Programmbeispiele.

Serielle Kommunikation mit Visual Basic

Dieses Online-Dokument macht Sie mit den Grundlagen der Datenfernübertragung vertraut und erläutert die serielle Programmierung unter Visual Basic anhand kleiner Programmbeispiele.

Serielle Kommunikation mit Visual Basic

Dieses Online-Dokument macht Sie mit den Grundlagen der Datenfernübertragung vertraut und erläutert die serielle Programmierung unter Visual Basic anhand kleiner Programmbeispiele.

Info

Dieser Online-Artikel beruht auf dem Beitrag **"Kommunikation mit Visual Basic for Windows"**, der in Ausgabe *3/94* der Fachzeitschrift Basic Professionell erschien. Außerdem kommt weiteres Material des Autoren zur Verwendung. Die HLP-Datei wurde mit dem Windows Help Magician erstellt.

Über den Autoren

ZOSCHKE DATA GmbH
Bahnhofstraße 3
D-24217 Schönberg/Holstein

Tel. 04344/6166 Fax: 04344/6162

CompuServe EMail: 71340,2051

ZOSCHKE DATA ist deutscher Exklusiv-Distributor der US-Tool-Hersteller Apex Software, Crescent Division of Progress, Desaware, Marquis Computing (AJS), Sheridan Software und Software Interphase. Ausführliche Produktinformationen werden auf Anfrage gerne zugesandt.

Help Magician 3.0 ist ein komfortables und komplettes System zum Erstellen von Windows-Hilfedateien für beliebige Applikationen und für Electronic Publishing. Die WYSIWYG-Umgebung der neuen Version 3.0 vermittelt das Gefühl, direkt in Winhelp zu arbeiten und erlaubt kontinuierliches Testen ohne zu kompilieren. Das Wichtigste in Kürze:

- Arbeitet Textverarbeitungs-unabhängig (Word etc. nicht erforderlich)
- Einfaches Einrichten von Verzweigungen und Popups, Auswählen von Browse-Folgen etc.
- Wizards für Glossar-Erstellung u. a.
- Unterstützung für Multimedia (WAV, AVI etc.) und Winhelp-Makros
- Umwandung von Handbüchern in Hilfe und umgekehrt
- VB-Quelltext-Scanner (erstellt Hilfe-Gerüst, in das nur noch Texte und Bilder einzufügen sind)
- Help Compiler und SHED.EXE enthalten
- Deutsches Handbuch, deutscher Support und Update-Service
- Update-Möglichkeit für Besitzer der Help-Magician-Version 2.x

Exklusiv-Distributor für Deutschland, Österreich und Schweiz: ZOSCHKE DATA GmbH

Einführung

Auch im Computerbereich bedeutet **Kommunikation** den *Austausch von Informationen*. Alle IBM-kompatiblen PCs sind mit zwei Typen von E/A-Kanälen (Ein-/Ausgabe) zur Kommunikation mit externen Geräten ausgestattet. Dies sind die seriellen und parallelen Schnittstellen; oft werden sie auch serielle und parallele *Ports* genannt.

Ein Parallel-Port wird als *Byte-orientiert* bezeichnet, da hier stets gleichzeitig die Übertragung von acht Bits über acht separate Leitungen geschieht. Auf diese Weise können Daten sehr schnell übertragen werden; aufgrund der Vielzahl der Drähte ist ein Kabel hierfür relativ teuer.

Einen seriellen Port ("COM-Port") nennt man *Bit-orientiert*, da hier Bit für Bit über ein und denselben Draht gesendet bzw. empfangen wird. Während die Übertragung zwar erheblich länger dauert als bei einer parallelen Schnittstelle, kommt man mit sehr wenigen Leitungen aus; dies fällt besonders bei langen Übertragungsstrecken ins Gewicht. Im Prinzip werden für eine Zweiweg- (Voll-Duplex-)Übertragung nur drei Leitungen benötigt - eine zum Senden, eine zum Empfangen und eine gemeinsame Masseleitung.

Unter Windows kommt den seriellen Schnittstellen eine besondere Bedeutung zu, denn zumindest eine wird zum Anschluß der unverzichtbaren Maus benötigt. In der Regel verfügt ein moderner PC über zumindest einen weiteren COM-Port, über den ein Modem, ein anderer PC oder zum Beispiel ein entsprechend ausgerüstetes Meßgerät angeschlossen werden kann. **Die seriellen Schnittstellen ermöglichen also ein breites Spektrum an Kommunikationsmöglichkeiten - auch mit Visual Basic für Windows!**

Für den erfolgreichen Einsatz der seriellen Schnittstellen - auch unter Visual Basic für Windows - sollte man ein wenig über deren Grundlagen wissen. Bevor wir uns in die Tiefen der Kommunikationsprogrammierung mit VB stürzen, hier also zunächst ein wenig Grundwissen.

Synchrone und asynchrone Kommunikation

Bei der seriellen Kommunikation wird zwischen **synchronem und asynchronem Betrieb** unterschieden. Bei der synchronen Kommunikation stimmen sich zwei Geräte zunächst aufeinander ab - sie synchronisieren sich. Anschließend senden sie kontinuierlich Zeichen, um im Takt zu bleiben. Selbst wenn gerade keine tatsächlichen Daten übertragen werden, weiß jedes Gerät anhand des kontinuierlichen Datenflusses über den Status des anderen Bescheid.

Das bedeutet, daß es sich bei den laufend übertragenen Zeichen entweder um "echte" Daten oder Füllzeichen zur Überbrückung von Pausen handelt. Synchroner Betrieb ermöglicht eine schnellere Übertragung als asynchroner Betrieb, da keinerlei zusätzliche Bits zur Markierung benötigt werden, wo ein Byte beginnt und wo es endet. Bei den seriellen PC-Ports handelt es sich jedoch nicht um synchrone, sondern um asynchrone Schnittstellen.

Der IBM PC und Kompatible arbeiten also mit asynchroner serieller Kommunikation. Asynchron bedeutet "keine Synchronisation". Diese Übertragungsart arbeitet somit ohne Füllzeichen; dafür muß jedoch für jeden Datenblock mit Hilfe von **Start-** und **Stop-Bits** angegeben werden, wann die Übertragung eines Datenblocks beginnt. Durch diese mindestens zwei zusätzlichen Bits ist eine asynchrone Übertragung langsamer als eine synchrone.

Eine asynchrone Leitung, die gerade nicht aktiv ist, wird durch den Wert 1 gekennzeichnet, was auch als *Mark*-Status bezeichnet wird. Durch Verwendung des Wertes 1 zur Anzeige, daß gerade keine Daten gesendet werden, können Geräte zwischen inaktivem Zustand und einer unterbrochenen Leitung unterscheiden. Steht ein Zeichen zum Senden an, wird dies durch ein Start-Bit signalisiert. Ein Start-Bit hat den Wert 0, was auch als *Space*-Status bezeichnet wird. Schaltet also eine Leitung von 1 auf 0, ist somit der Empfänger alarmiert, daß ein Datenzeichen folgt.

Übertragung Bit für Bit

Nachdem ein Start-Bit gesendet wurde, schickt der Sender die tatsächlichen Daten-Bits. Je nach gewählter Einstellung können dies 5, 6, 7 oder 8 Bits sein. Sender und Empfänger müssen sich natürlich nicht nur über das Timing der Start- und Stop-Bits, sondern auch über die Anzahl der dazwischenliegenden Daten-Bits einig sein, d.h. mit denselben **Übertragungs-Parametern** arbeiten.

Beachten Sie, daß bei Verwendung von nur 7 Bits keine ASCII-Werte größer als 127 übertragen werden können; entsprechend ist der höchste Wert bei 5 Bits 32. Dies ist besonders für Binär-Daten (z.B. Programme oder Bilder) von Bedeutung, sowie für Texte mit fremdsprachlichen Sonderzeichen (sämtliche deutschen Umlaute haben einen höheren ASCII-Code als 127). Nachdem die Daten-Bits übertragen sind, wird ein Stop-Bit hinterhergeschickt. Ein Stop-Bit hat den Wert 1 (Mark) und kann sogar dann erkannt werden, wenn das zuletzt gesendete Daten-Bit ebenfalls 1 war. Erreicht wird dies durch die zeitliche Länge des Stop-Bits; sie kann wahlweise 1, 1,5 oder 2 Bit-Perioden betragen.

Das Paritäts-Bit

Außer den Start-, Stop- und Daten-Bits wird wahlweise ein weiteres Bit übertragen, das als **Paritäts-Bit** bezeichnet wird. Ein solches Bit bietet in bescheidenem Umfang eine Fehlerprüfung, indem eine Veränderung der Daten während der Übertragung angezeigt wird. Hier kann zwischen gerader, ungerader oder keiner Paritätsprüfung gewählt werden (*Even, Odd* oder *No Parity*). Wird mit Paritätsprüfung gearbeitet, wird die Anzahl der Marks (Bits mit logischem Pegel 1) gezählt. Ein einzelnes Bit, das im Anschluß an die Daten-Bits übertragen wird, gibt dann an, ob die Anzahl der gesendeten 1er-Bits gerad- oder ungeradzahlig war. Wurde zum Beispiel eine gerade Parität (even) gewählt, wird ein Paritäts-Bit mit dem Wert 0 übertragen, wenn die Anzahl der zuvor übertragenen Daten-Bits geradzahlig ist. Für den Binärwert 0110 0011 wäre die Parität also 0, bei 1101 0110 wäre sie 1. Bei ungerader Parität funktioniert dies genau umgekehrt: Das Paritäts-Bit ist 0, wenn die Anzahl der Mark-Bits im vorangegangenen Wort einen ungeraden Wert ergibt.

Es ist wichtig zu verstehen, daß die Paritätsprüfung nur eine sehr grobe Fehlerdiagnose liefert. Zum einen gibt sie nicht an, welches Bit eines Wortes fehlerhaft ist, zum anderen versagt sie, wenn in einem Wort zwei 1er-Bits fehlerhaft übertragen werden und die Fehler sich somit aufheben.

Bidirektionale Kommunikation

Nun zu zwei weiteren wichtigen Begriffen. Der erste ist **Simplex**-Kommunikation; bei dieser Methode können die Daten nur in einer Richtung reisen. Dies ist die einfachste Art der Datenübertragung, wobei ein PC oder Terminal ausschließlich als Empfänger, das andere Gerät ausschließlich als Sender fungiert. Ein typisches Beispiel für eine Simplex-Übertragung ist die parallele Drucker-Schnittstelle; hier fließen die Daten nur vom PC zum Drucker. **Halbduplex-Betrieb** hingegen beschreibt eine Verbindung, in der beide Geräte sowohl senden als auch empfangen können, allerdings nicht gleichzeitig. Einige der weiter hinten beschriebenen Modem-Protokolle sind auf Halbduplex-Betrieb ausgelegt. Demgegenüber bedeutet **Vollduplex**, daß beide Geräte gleichzeitig senden und empfangen können. Die seriellen PC-Schnittstellen sind auf Vollduplex-Betrieb ausgelegt. Es sind separate Leitungen zum Senden und zum Empfangen von Daten vorhanden; einige weiter hinten besprochenen Übertragungs-Protokolle unterstützen Vollduplex-Übertragung auf einer einzigen Leitung.

Bits pro Sekunde = Baud?

Einer der am häufigsten mißbrauchten Begriffe der seriellen Datenübertragung ist **Baud**, was landläufig mit "Bits pro Sekunde" (BPS) gleichgesetzt wird.

Die Einheit Baud wurde nach *Jean Maurice Emile Baudot*, einem Mitarbeiter des Französischen Telegrafen-Service benannt. Ihm gebührt der Ruhm für die erstmalige Verwendung eines 5 Bit-Codes für Buchstaben Ende des 19. Jahrhunderts. Worauf sich die Einheit "Baud" wirklich bezieht, ist die Modulationsrate. Sie gibt an, wie oft pro Sekunde eine Leitung ihren Pegel wechselt. Ist dies nicht dasselbe wie BPS? Nicht unbedingt! Verbinden Sie zwei serielle Ports durch direkte Verkabelung, stimmen Baud und BPS in der Tat überein. Läuft die Übertragung zum Beispiel mit 38400 BPS, wechselt auch die Leitung 38400-mal ihren Pegel. Bei der Verwendung von Modems ist dies jedoch nicht der Fall (ein Modem ist das Bindeglied zwischen Telefonleitung und Computer zur Umwandlung der seriellen Bits (High-/Low-Pegel) in Töne (Frequenzen) und umgekehrt. Das Kunstwort ist eine Kombination der Begriffe Modulator/Demodulator).

Da Modems Signale über die Telefonleitung übertragen, ist die maximale Baudrate ziemlich begrenzt. Je nach Modulationstyp liegt diese Grenze bei 1200 oder 2400 Baud. Dies ist eine physikalische Begrenzung, die durch die Leitungstechnik der jeweiligen Telefongesellschaft vorgegeben ist. Wenn Sie beispielsweise meinen, ein 2400-Baud-Modem zu haben, kann es sein, daß dieses Gerät tatsächlich mit 9600 BPS oder höher übertragen kann. In der Tat sind 2400 BPS-Modems eigentlich 600-Baud-Geräte, wobei der höhere Datendurchsatz durch ausgeklügelte Phasenmodulations-Techniken erzielt wird.

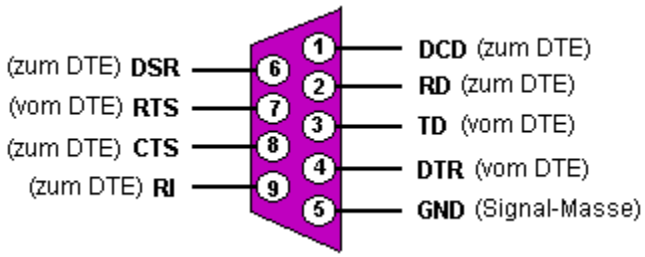
Der RS-232C-Standard

Ein Begriff, der in Zusammenhang mit der seriellen Datenübertragung häufig auftaucht, ist **RS-232C**. Das "RS" steht dabei für *Recommended Standard* (= Empfohlener Standard), das "C" gibt die Version an. Die serielle Schnittstelle der meisten Computer ist eine Untermenge des RS-232C-Standards. Der komplette RS-232C-Standard beschreibt einen 25-poligen "D"-Steckverbinder, bei dem 22 Kontakte belegt sind. Die meisten dieser Kontakte werden für die PC-Kommunikation nicht benötigt; in der Praxis sind viele neuere Computer mit einem reduzierten 9-poligen seriellen Steckverbinder ausgestattet, wie er mit dem PC/AT eingeführt wurde. Zur Anpassung an 25-polige Steckverbinder gibt es Übergangsadapter.

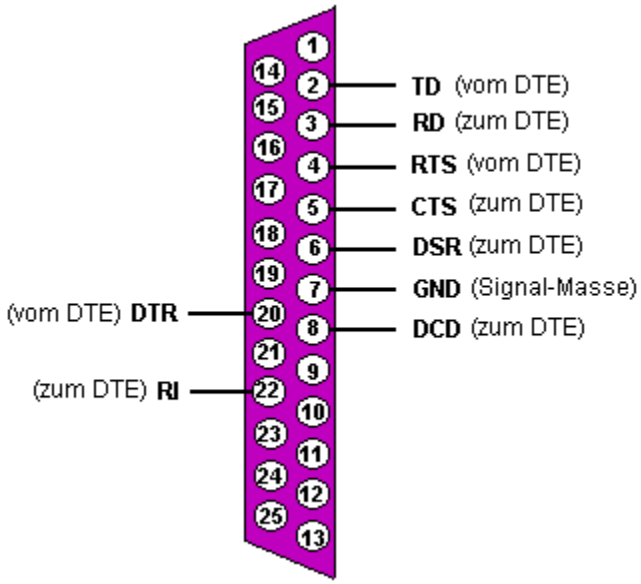
9-poliger Steckverbinder

25-poliger Steckverbinder

Vergleichstabelle



9-poliger D-Steckverbinder



25-*poliger D-Steckverbinder*

DCE und DTE bestimmen die Richtung

Zwei weitere Begriffe, die Sie kennen sollten, sind **DTE** und **DCE**. DTE steht für *Data Terminal Equipment*, DCE für *Data Communications Equipment*. RS-232-Leitungen sind nicht bidirektional; darum wird mit diesen Begriffen definiert, in welche Richtung Daten fließen. Ein Computer ist üblicherweise ein DTE-Gerät, ein Modem ein DCE-Gerät. Die meisten Geräte lassen sich wahlweise als DTE oder DCE konfigurieren.

Der RS-232-Standard gibt an, daß DTE-Geräte einen 25-poligen Stift-Steckverbinder (male, "Männchen"), DCE-Geräte einen entsprechenden Buchsen-Steckverbinder (female, "Weibchen") verwenden sollen. Daher läßt sich ein DTE- mit einem DCE-Gerät unter Verwendung eines Kabels koppeln, in dem einfach Kontakt für Kontakt durchverdrahtet ist (Eins-zu-Eins-Verdrahtung). Sollen hingegen zwei gleichartige Geräte miteinander verbunden werden, ist hierzu ein Nullmodem-Kabel erforderlich. In einem solchen Kabel werden die Sende- und Empfangsleitungen kreuzweise vertauscht; weiter hinten kommen wir auf dieses Thema noch zu sprechen. Beachten Sie, daß im weiteren Verlauf der Erläuterungen mit DTE Ihr Computer gemeint ist.

Für die Programmierung kann es wichtig sein, die Bedeutung der einzelnen Leitungen des seriellen Verbindungskabels zu kennen; daher wollen wir sie etwas näher betrachten. Die Leitung **TD** (*Transmit Data*, Daten senden) ist diejenige, über die ein DTE-Gerät Daten sendet. Diese Bezeichnung kann Verwirrung stiften, da dieselbe Leitung vom DCE zum Empfang der Daten verwendet wird. Findet auf dieser Leitung gerade keine Übertragung statt, wird sie vom DTE-Gerät auf Mark-Pegel gehalten. Die Leitung **RD** (*Receive Data*, Daten empfangen) ist diejenige, über die ein DTE-Gerät Daten empfängt; findet auf dieser Leitung gerade keine Übertragung statt, wird sie vom DCE-Gerät auf Mark-Pegel gehalten.

Steuerleitungen

Neben den Leitungen zum Senden und Empfangen stellt die serielle Schnittstelle eine Reihe von Steuerleitungen zur Verfügung; die am häufigsten verwendeten werden nachfolgend erläutert.

RTS (*Request To Send*)
DTR (*Data Terminal Ready*)
CD (*Carrier Detect)*
RI (Ring Indicator)

Zum Thema Leitungen sei angemerkt, daß der RS-232C-Standard eine maximale Kabellänge von ca. 15 m (50 ft.) vorschreibt. Dies kann man in den meisten Fällen jedoch getrost ignorieren; bei guten seriellen Schnittstellen-Karten und Baudraten bis zu 19200 stellt eine Kabellänge bis zu ca. 90 m (300 ft.) meist kein Problem dar. Besonders bei längeren RS232-Leitungen ist es jedoch erforderlich, qualitativ hochwertiges und gut abgeschirmtes Kabel zu verwenden.

RTS steht für *Request To Send* (Sende-Anfrage). Das DTE-Gerät setzt diese Leitung auf Space-Pegel, um dem anderen Gerät mitzuteilen, daß es zum Senden von Daten bereit ist. Das Gegenstück zu dieser Leitung ist CTS (Clear To Send, bereit zum Senden). Das DCE-Gerät setzt diese Leitung auf Space-Pegel, um dem DTE- Gerät mitzuteilen, daß es zum Empfang von Daten bereit ist. Zusammen führen diese Leitungen einen Quittungsbetrieb durch, der als CTS/RTS-Handshake bezeichnet wird. Das zu einem späteren Zeitpunkt erläuterte MSComm-Control unterstützt sowohl diese Art von Hardware-Handshake als auch den XOn/XOff-Quittungsbetrieb mit Steuerzeichen (auch als Software-Handshake bezeichnet).

DTR steht für *Data Terminal Ready* (Datenterminal bereit). Diese Leitung funktioniert ähnlich wie RTS und dient zur Steuerung der Ausgabe eines DTE-Gerätes. DSR (Data Set Ready, Datensatz bereit) ist in derselben Weise das Gegenstück zu DTR wie CTS zu RTS. Der DTR/DSR-Handshake arbeitet im Prinzip genauso wie der CTS/RTS-Handshake; er wird vom weiter hinten behandelten VB-Kommnuikations-Control MSComm jedoch nicht unterstützt, da die meisten Modems DTR zur Unterbrechung einer Telefonverbindung verwenden. Zieht der Computer die DTR-Leitung auf False-Pegel, wird ein Modem damit in der Regel zum "Auflegen" bewegt. Dies erklärt auch eine unerfreuliche QuickBASIC-Eigenart: Schließt QB einen COM-Port, zieht es automatisch die DTR-Leitung auf False-Pegel. MSComm hingegen beläßt DTR in dem Zustand, in dem die Leitung bei Öffnen des Ports vorgefunden wurde, und gibt Ihnen beim Schließen Gelegenheit, den DTR-Status bei Bedarf selbst zu ändern.

CD steht für *Carrier Detect* (Trägersignal-Erkennung). Mit dieser Leitung signalisiert ein Modem, daß es Verbindung mit einem anderen Modem hat oder ein Trägersignal erkannt hat. Die CD-Leitung erweist sich als nützlich bei der Ermittlung, ob das Gerät am anderen Ende aufgelegt hat. Auf das Trägersignal wird im Abschnitt über Modems eingegangen.

RI steht für *Ring Indicator* (Läuten-Anzeiger). Ein Modem ändert den Pegel dieser Leitung, wenn aufgrund eines eingehenden Anrufs das Telefon läutet. Leider unterstützen jedoch nicht alle Modems und seriellen Schnittstellen, nicht einmal alle Kabel diese Leitung (generell ist bei allzu "preiswerten" Drucker- und Modemkabeln Skepsis angebracht; oft fehlen hier aus Kostengründen ein paar Adern).

Kabel, Null-Modems und Adapter

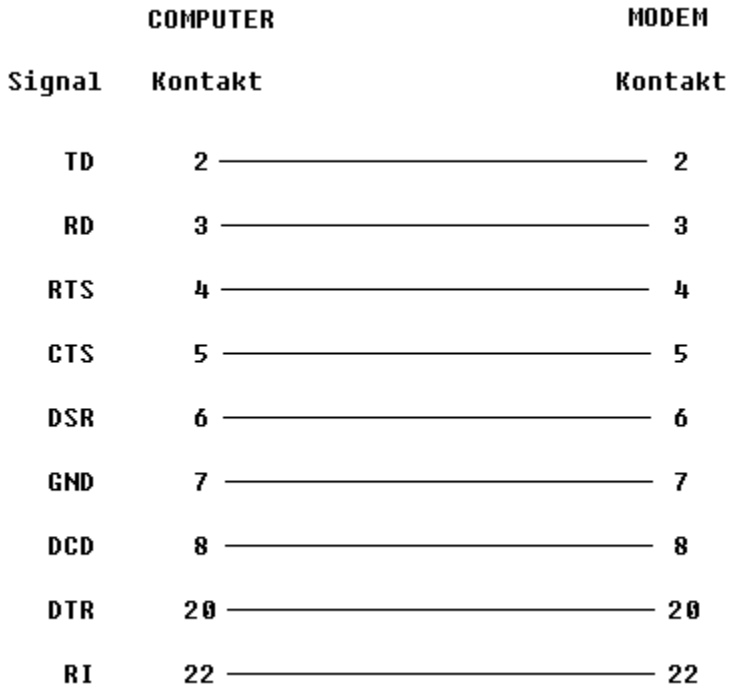
Wäre alles perfekt, gäbe es an allen DTE-Geräten ausschließlich 25-polige "D"-Steckverbinder mit Stiften ("Männchen"); alle Modems, Drucker und was sich sonst noch alles an einen Computer anschließen läßt, hätten dazu passend Buchsen-Steckverbinder ("Weibchen"), so daß man zur Verbindung einfach nur ein 1:1-verdrahtetes Kabel benötigen würde (Pin 1 des einen Endes mit Pin 1 des anderen verbunden, Pin 2 des einen mit Pin 2 des anderen usw.). Die Realität sieht leider anders aus. Es gibt 9- und 25-polige Steckverbinder, und viele Geräte lassen sich entweder als DTE oder als DCE konfigurieren. Aus diesem Grund gibt es Nullmodem-Kabel und Wechseladapter (*Gender Changer*, wörtlich "Geschlechtswandler").

Um zwei Computer seriell miteinander zu koppeln, gibt es im Prinzip zwei Möglichkeiten. Zum einen können zwei Computer mit Hilfe von Modems kommunizieren. Dies ist die einzige Möglichkeit zur seriellen Kommunikation, wenn die Computer physikalisch sehr weit voneinander entfernt stehen, im Extremfall in verschiedenen Kontinenten. Die Verbindung zwischen Computer und Modem mit einem Kabel mit 25-poligen Steckverbindern zeigt Ihnen die Abbildung; hier wurden alle wichtigen Steuerleitungen verbunden.

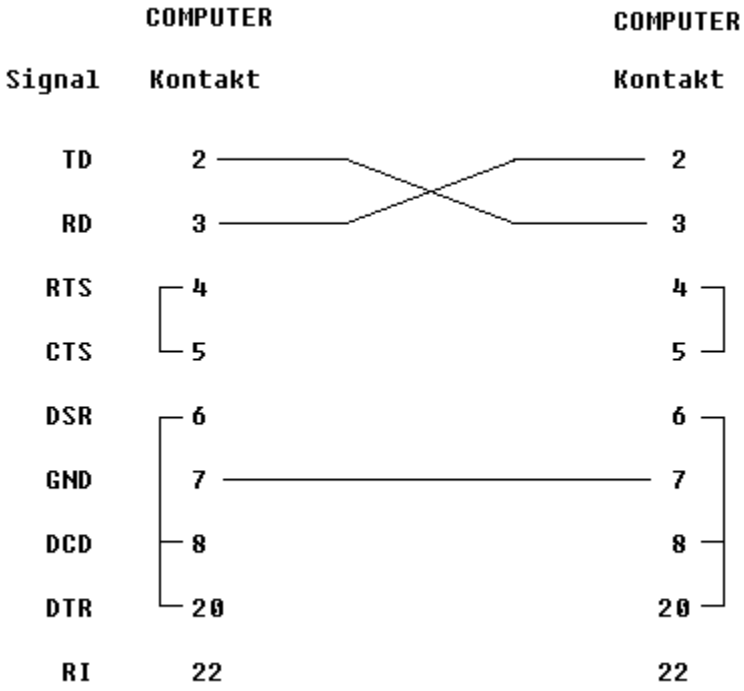
Stehen die Computer jedoch dicht beieinander, kann man die Geräte direkt miteinander verkabeln. Dies erspart nicht nur die Ausgabe für die Modems, sondern ermöglicht auch eine erheblich höhere Übertragungsgeschwindigkeit. Mit Hilfe eines sogenannten **Nullmodem-Kabels** lassen sich zwei DTE-Geräte direkt verbinden, ohne daß Modems erforderlich wären - daher der Name. Die Minimal-Ausführung eines Nullmodem-Kabels kommt mit nur drei Leitungen aus - kreuzweise wird die Sendeleitung des einen Geräts mit der Empfangsleitung des anderen verbunden; hinzu kommt eine gemeinsame Masseleitung. Die entsprechenden werden am jeweiligen Anschluß einfach gebrückt (siehe *Abbildung*). Für viele Zwecke ist eine solche Verbindung absolut ausreichend und beim Autoren seit mehr als 10 Jahren zur Rechnerkopplung erfolgreich im Einsatz. Mit dieser Konfiguration ist allerdings kein CTS/RTS- oder DSR/DTR-Quittungsbetrieb möglich. Soll ein solcher Hardware-Handshake stattfinden, müssen auch die hierfür zuständigen Leitungen über Kreuz durchverdrahtet werden.

Probleme können auftreten, wenn das eine Kabel mit 25-poligem und das andere mit 9-poligem Steckverbinder ausgestattet ist. In diesem Fall muß ein Adapter angefertigt oder gekauft werden. Bei vielen Schnittstellenkarten mit 9-poligem Steckverbinder sowie Mäusen und andere seriellen Zusatzgeräten ist ein Adapter zur Anpassung an 25-polige Schnittstellen im Lieferumfang enthalten. Falls Sie keinen solchen Adapter haben, aber mit einem Lötkolben umgehen können, hilft Ihnen die Gegenüberstellung der Anschlußbelegungen in der *Tabelle*. Ein Standard-Adapter besteht aus einem 9-poligen Buchsen- und einem 25-poligen Stift-Steckverbinder; es sind natürlich auch andere Kombinationen möglich.

Ein weiteres Problem, auf das Sie stoßen könnten, wären zwei Steckverbinder gleichen "Geschlechts". Auch für diesen Fall gibt es fertige Adapter ("Gender Changer") im Zubehör-Handel zu kaufen, oder man fertigt sie selbst an. Beispielsweise können Sie mit einem Adapter, der an beiden Enden Stift-Steckverbinder hat (1:1 durchverdrahtet), zwei Geräte bzw. Kabel mit Buchsen-Steckverbindern verbinden.



Computer-Modem-Verbindung



Nullmodem-Kabel mit drei Leitungen

Modems

Mit Hilfe von **Modems** lassen sich Computer per Telefonleitung über große Entfernungen verbinden. "Modem" ist ein Kustwort, das sich aus Modulator und Demodulator zusammensetzt. Kurz gesagt dient diese Einheit dazu, die logischen High/Low-Datenpegel zur Telefonübertragung in Tonfrequenzen umzuwandeln ("Modulieren") und am anderen Ende wieder in High/Low-Datenpegel ("Demodulieren"). Wie unter Bits pro Sekunde=Baud? beschrieben, sind der Übertragungsgeschwindigkeit durch die Verwendung des Telefon-Frequenzbereichs Grenzen gesetzt.

In den Anfangstagen des PC, als man 64 K Speicher für völlig ausreichend hielt, war auch ein 300 BPS-Modem Stand der Technik. Falls Sie jemals mit einem 300 BPS-Modem gearbeitet haben, erinnern Sie sich sicher, wie langsam und frustrierend Datenübertragung mit dieser Geschwindigkeit ist. Glücklicherweise sind mittlerweile Modems verfügbar, die mehr als 32-mal so schnell arbeiten.

Modems mit 2400 BPS sind inzwischen längst keine teuren Exoten mehr und werden vielerorts mit und ohne ZZF-Nummer ("nur für Export") extrem preiswert angeboten. Um potentiellem Ärger aus dem Weg zu gehen, sollte man auf ein postalisch zugelassenes Modem zurückgreifen. Auch Modems mit 9600 und 14400 BPS gewinnen zunehmend an Popularität; die sinnvolle Grenze liegt momentan bei 38400 BPS, wobei eine ganze Reihe ausgeklügelter Techniken wie beispielsweise Datenkompression zum Einsatz kommt. Die *Tabelle* enthält eine Aufstellung gängiger Modem-Standards.

Falls Sie die Anschaffung eines Modems nach dem neuesten Stand der Technik planen, sollten Sie sich nach einem V.32-Modem umsehen, das mit V.42-Fehlerkorrektur und V.42b-Datenkompression arbeitet. Damit ist sichergestellt, daß Ihr Modem auch mit den meisten zukünftigen Hochgeschwindigkeits-Modems kommunizieren kann. Aufgrund der steigenden Zahl der Hersteller solcher Modems ist auch mit einem stetigen Sinken der Preise zu rechnen.

Fast alle Modems werden über eine Reihe spezieller Befehle gesteuert, die allgemein als Hayes-"AT"-Befehle bezeichnet werden, da Hayes die Zeichen "AT" (*Attention*, Achtung) als Präfix-String gewählt hat, der einem Modem signalisiert, daß anschließend ein Befehl folgt. Welche AT-Befehle Ihr Modem unterstützt, wird in der Regel im Anhang des zugehörigen Handbuchs beschrieben; auch das Handbuch zu PDQComm for Windows behandelt alle gängigen AT-Befehle.

UARTs

Das Kunstwort **UART** steht für *Universal Asynchronous Receiver Transmitter* (Universeller Asynchroner Empfänger/Sender). Ein UART-Baustein ist das Herz einer jeden COM-Schnittstelle; man kann ihn sich vereinfacht als Parallel-/Seriell-Wandler vorstellen. Der Baustein nimmt von der CPU ein Byte zur Zeit entgegen und wandelt es in einen seriellen Datenstrom, der dann gesendet wird. Umgekehrt empfängt der UART serielle Bits und wandelt sie in Bytes zurück, mit denen die CPU etwas anfangen kann. Neben dieser Datenwandlung steuert der UART auch die diversen Steuerleitungen, die für den Quittungs-Betrieb benötigt werden.

Für Windows-Benutzer ist es wichtig zu wissen, daß in PC-Schnittstellen unterschiedliche UART-Typen zum Einsatz kommen.

In den IBM PCs bzw. ATs der ersten Generation wurden UARTs vom Typ INS8250 bzw. NS16450 eingesetzt; da dieses Bauteil sehr billig ist, findet es noch heute in der Mehrzahl der seriellen Schnittstellen Verwendung. Der hardwaremäßig vorgesehene Empfangspuffer enthält bei diesem Typ nur das zuletzt über die serielle Leitung eingetroffene Zeichen; auch der Sendepuffer - hier werden zu sendende Bytes zwischengespeichert - kann nur ein Zeichen aufnehmen.

In hochwertigen Schnittstellenkarten der neueren Generation sowie in Hochgeschwindigkeits-Modems kommen UART-Bausteine mit der Bezeichnung NS16550A zum Einsatz. Kleine Verbesserung mit großer Wirkung: NS16550A enthält sowohl zum Senden als auch zum Empfangen jeweils einen FIFO-Puffer (First In First Out) von jeweils 16 Bytes. Dies bewirkt eine Leistungssteigerung der seriellen Übertragung, da mehrere Bytes gespeichert werden können, bevor ein Interrupt erzeugt wird - die CPU wird somit seltener in ihrem sonstigen Arbeitsablauf unterbrochen. Besonders wichtig ist dies beim Einsatz von Multitasking-Betriebssystemen wie Microsoft Windows. Besonders höheren Übertragungsraten werden durch Einsatz von COM-Schnittellen mit NS16550A (oder dazu kompatiblen Chips) sicherer oder überhaupt erst möglich. Ein Austausch der Schnittstellenkarten kann sich daher lohnen. Beachten Sie bei einem Austausch, daß auf neuen Karten eventuell Brücken (Jumper) zur Zuweisung der Ports (COM 1-4) und der zugehörigen IRQ-Interruptnummern eingestellt werden müssen.

Auf der Heftdiskette zur Basic Professionell (Ausgabe 3/94) finden Sie eine kleine DOS-Utility, mit deren Hilfe Sie den Typ der in Ihrem PC installierten UARTs ermitteln können (GETUART.EXE).

Dateiübertragungsprotokolle

Mit MSComm.VBX ist es möglich, Daten-Bytes zu senden und zu empfangen; prinzipiell lassen sich daher bereits mit diesem VB-Control auch Dateien senden und empfangen. Allerdings wird hierbei keinerlei Fehlerkorrektur vorgenommen. In erster Linie ist diese Methode für Textdateien geeignet. Fehlt hier einmal ein Zeichen oder wird es fehlerhaft übertragen, tut dies der Lesbarkeit in der Regel keinen Abbruch und kann auf der Empfangsseite leicht korrigiert werden. In einer gepackten ZIP-Datei beispielsweise führt aber bereits ein einziges fehlerhaft übertragenes Bit dazu, daß sich die Datei nicht mehr korrekt entpacken läßt. Für Binärdateien wir Bilder, Programme und Archivdateien sollte daher ein sogenanntes Übertragungsprotokoll verwendet werden, wie es PDQComm for Windows zur Verfügung stellt.

Durch ein solches Protokoll einigen sich beide Seiten auf Maßnahmen zur Fehlerprüfung und -Korrektur. Sehen wir uns als Beispiel das XModem-Protokoll an: Nacheinander wird hier jeweils ein Datenblock von 128 Bytes und daran anschließend dessen Prüfsumme übertragen; der Empfänger rechnet für jeden erhaltenen Block dessen Prüfsumme nach und fordert beim Sender im Falle eines Fehlers die Wiederholung des betroffenen Blocks an. Je nach verwendetem Fehlerprüfverfahren wird zwischen XModem Checksum und XModem CRC unterschieden.

XModem ist eines der ältesten Übertragungsprotokolle für serielle Datenübertragung. Aufgrund seiner Verbreitung wird es auch noch heute vielfach eingesetzt; auch im Windows-eigenen Terminalprogramm ist es verfügbar. Inzwischen gibt es jedoch eine Reih weiter fortgeschrittener Protokolle, die durch mehr "Intelligenz" für mehr Effizienz und Datensicherheit sorgen; einige erlauben auch Stapelverarbeitung, automatischen Übertragungsstart und vieles mehr. Hier eine Kurzcharakteristik der gängigsten Protokolle, die auch allesamt vom PDQComm-Control unterstützt werden:

XModem-1K

YModem

YModem Batch

ZModem

Compuserve B+

Kermit

XModem-1K ist eine XModem-Variante, bei der Blöcke von 1024 Bytes übertragen werden. Dadurch ist die Gesamtübertragungszeit kürzer als unter Verwendung des XModem-Protokolls.

YModem ist eine XModem-Variante, die mit Blöcken von 1024 Bytes und CRC arbeitet.

YModem Batch ist eine YModem-Variante zur Übertragung einer oder mehrer Dateien mit einem einzigen Befehl.

ZModem ist eine XModem-Variante, die variabel lange Datenblöcke verwendet. Werden Übertragungsprobleme festgestellt, so werden die Blöcke automatisch verkürzt.

Compuserve B+ ist ein Protokoll, das von CompuServe speziell für deren weltweit erreichbaren Online-Informationsdienst entwickelt wurde. Es überträgt nicht nur schnell, sondern wird auch recht gut mit schlechten Telefonleitungen fertig.

Kermit ist ein besonders bei der Minicomputer- und Mainframe-Kopplung beliebtes Protokoll. Es verwendet kurze Datenblöcke und kann unter Verwendung von 7-Bit-Zeichen 8-Bit-Daten übertragen. Kermit ist zwar langsam, dank diverser Selbstjustierungsfähigkeiten jedoch recht flexibel. Obwohl ebenfalls schon etwas betagt, bietet es Features wie Neusynchronisation bei schlechter Leitung, Wildcard-Übertragung (z. B. alle *.PCX) und Datenkompression. Neben XModem ist auch Kermit im Windows-Terminalprogramm verwendbar.

Was ist anders bei Windows?

Die geschilderten Grundlagen gelten gleichermaßen für serielle Kommunikation unter DOS und unter Windows. Obwohl die momentane Windows-Version 3.x auf DOS aufsetzt, muß die Kommunikation in diesen beiden Umgebungen unterschiedlich gehandhabt werden. Hier die die zwei wichtigsten Unterschiede:

Windows ist eine Multitasking-Umgebung - Ihr Programm hat daher nicht mehr die alleinige Kontrolle über den Computer und seine Schnittstellen, sondern muß die Rechenzeit mit anderen Anwendungen "teilen".

Windows hat seine eigene Schnittstelle für Anwendungsprogramme (API, Application Programmer's Interface), die sich gänzlich von allem unterscheidet, was DOS bietet. Während man unter DOS durch direktes Schreiben in die UART-Register das Letzte aus der COM-Schnittstelle herausholen kann (mit QuickBASIC und Assembler-Zusatzbibliotheken bis zu 115200 Baud), muß der Zugriff bei Windows 3.x über das API erfolgen (die API-Funktionen für Kommunikation befinden sich bei Windows im Treiber COMM.DRV). Dazu gleich noch mehr.

Geräte-Unabhängigkeit

Unter DOS gibt es im Prinzip drei verschiedene Möglichkeiten, um auf die COM-Ports zuzugreifen:

Auf der höchsten Ebene kann **DOS** dazu benutzt werden, den COM-Port wie eine Datei zu öffnen. Das folgende Code-Beispiel würde per Modem die Basic_Box des Steingäber Fachverlags anwählen:

```
Open "Com2" For Output As #1
Wahl$ = "0431-737121"
Print #1, Wahl$
Close #1
```

Leider sind die von DOS zur Verfügung gestellten Möglichkeiten sehr beschränkt. Zwar können Bytes von einem COM-Port gelesen und dorthin geschrieben werden; es gibt jedoch keine Möglichkeit zur Kontrolle wichtiger Kommunikations-Einstellungen wie Geschwindigkeit, Anzahl der Start- und Stop-Bits oder der Parität.

Um dies zu ermöglichen, ist es erforderlich, eine Ebene tiefer zu gehen und das **BIOS** (Basic Input Output System) des Computers zu verwenden. Das BIOS ist ein kleines Programm, das sich im ROM Ihres Computers befindet und anderen Programmen Service-Routinen zum Zugriff auf die Hardware zur Verfügung stellt. Die BIOS-Kommunikations-Services bieten mehr Kontrolle, lassen aber noch immer eine wichtige Möglichkeit missen, nämlich die gepufferte Ein- und Ausgabe. Selbst wenn nur wenige Zeichen am Port eintreffen, während Ihr Programm gerade mit etwas anderem beschäftigt ist, gehen diese ohne Zwischenspeicherung in einem Puffer verloren.

Die niedrigste Ebene des Zugriffs auf den COM-Port ist zugleich die effizienteste. Durch **direktes Schreiben** in die Register und Ports des Kommunikations-Steuerbausteins lassen sich gepufferte Ein-/Ausgabe und alle nur denkbaren Optionen ermitteln und einstellen. Nahezu sämtliche heute kommerziell erhältliche Kommunikations-Software (einschließlich Tool-Bibliotheken wie PDQComm/DOS für QuickBASIC, BASIC PDS und VB/DOS) führt auf diese Weise die Kommunikation durch.

Das Windows-Kommunikations-API

Da Windows dem System noch eine weitere Schicht hinzufügt, könnte man vermuten, daß die hier verfügbaren Funktionen noch unbrauchbarer wären als die DOS-Kommunikations-Services. Dies ist zum Glück nicht der Fall. Das Windows-Kommunikations-API ist recht leistungsfähig und bietet nicht nur umfangreiche Möglichkeiten zur Port-Kontrolle, sondern ermöglicht auch gepuffertes Senden und Empfangen. Bisher war bereits mehrmals von Puffern und Pufferung die Rede. Es sei angemerkt, daß ein "Puffer" einfach ein Zwischenspeicher ist. Da das Senden und Empfangen oft schneller geht als die Software die Daten verarbeiten kann, werden sie zwischengespeichert, eben "gepuffert". Neben den angesprochenen Hardware-Puffern, die in die seriellen Übertragungsbausteine eingebaut sind, lassen sich per Software Speicherbereiche im Arbeitsspeicher des Rechners resevieren; dies wird als Software-Pufferung bezeichnet.

Wie nahezu jede Funktion in Windows werden auch die Kommunikations-Funktionen von Treibern zur Verfügung gestellt. Liefert ein Hardware-Hersteller einen völlig anderen Kommunikations-Chip (z.B. mit einer Mehrfachschnittstellen-Karte), muß er daher lediglich einen speziellen Treiber (*Device Driver*) dazu schreiben, damit *alle* "sauber" geschriebenen Windows-Kommunikations-Programme mit der neuen Hardware arbeiten können. Wird also ausschließlich das Windows-Kommunikations-API verwendet, hat dies zur Folge, daß Ihre Windows-Programme völlig geräteunabhängig (*device independent*) sind. Da diese vorteilhafte Technik wird auch von dem in der *Visual Basic 2.0/3.0 Professional Edition* enthaltenen Kommunikations-Control *MSComm* (und dazu kompatiblen Zusatz-Tools) genutzt wird, sind auch Ihre mit VB entwickelten Windows-Kommunikationsanwendungen geräteunabhängig und universell einsetzbar!

Multitasking

Da Windows eine Multitasking-Umgebung ist, kann eine ganze Menge passieren, während ein Programm läuft. Zum Beispiel könnte ein Tabellenkalkulations-Formular gerade Berechnungen durchführen, ein Datenbank-Programm eine größere Datei sortieren oder der Druckmanager eine Datei ausdrucken. All diese Dinge könnten Ihrem Programm verhältnismäßig viel Prozessorzeit stehlen. Daher ist es so außerordentlich wichtig, mit gepufferter Ein-/Ausgabe zu arbeiten: Treffen Zeichen ein, während Windows sich gerade einer anderen laufenden Anwendung widmet, werden diese zwischenzeitlich in einem "Puffer" untergebracht, auf den später zugegriffen werden kann. Microsoft Windows 3.x ist ein sogenanntes nicht-preemptives Multitasking-System. Das bedeutet, daß das Betriebsystem nicht einfach Ihrem Programm die Kontrolle entreißt und an ein anderes Programm übergibt. Vielmehr muß Ihr Programm die Kontrolle an Windows übergeben. In Visual Basic kann dies mit Hilfe der *DoEvents*-Funktion geschehen. Desweiteren ist anzumerken, daß auf einen bestimmten Kommunikations-Port jeweils nur von einem Programm zur Zeit zugegriffen werden kann. Ein COM-Port sollte daher nur dann geöffnet werden, wenn er benötigt wird. Ist die Kommunikation abgeschlossen, sollte auch der Port gleich wieder geschlossen werden, um ihn anderen Windows-Programmen zur Verfügung zu stellen.

Beim Programmieren in Visual Basic für Windows brauchen Sie selbst praktisch keinerlei Aufwand zu betreiben, um ein Programm mit der Fähigkeit zur Hintergrund-Kommunikation auszustatten. Alles, was dazu von Ihrer Seite noch zu erledigen ist, ist gegebenenfalls das Einfügen einiger *DoEvents*-Aufrufe; den Rest erledigt Windows für Sie.

VB und das Kommunikations-API

Wie bereits erwähnt, geschieht das Senden und Empfangen serieller Daten unter Windows durch den Aufruf von API-Funktionen, die im installierten Kommunikationstreiber (COMM.DRV) enthalten sind. Die eigentliche Arbeit der UART-Steuerung übernehmen die Treiber-Routinen.

Auch mit Visual Basic für Windows muß man sich an diese Spielregeln halten. Für einfachere Aufgaben ist es sogar möglich, die API-Funktionen zu deklarieren und direkt aufzurufen.

Wie die *Tabelle* zeigt, unterscheiden sich die Kommunikations-API-Befehle garnicht allzusehr von VB-Ein-/Ausgabe-Anweisungen, wie man sie zum Beispiel zum Lesen und Schreiben von Dateien benutzt. Eine Aufstellung aller verfügbaren Kommunikations-API-Funktionen und der von ihnen benötigten Type-Variablen finden Sie im mit Visual Basic (Professional Edition) gelieferten Hilfedatei WIN31API.HLP bzw. in der Datei WINAPI.TXT.

Die Programmierung komplexerer Kommunikations-Anwendungen unter direkter Nutzung von API-Funktionen erfordert allerdings erhebliche Klimmzüge und kommt meist nicht ohne fremde Hilfsmittel aus. Das in [2] vorgestellte einfache Kommunikationsprogramm nutzt zum Beispiel das auf der zugehörigen Buchdiskette mitgelieferte Callback-Control. Das in [3] enthaltene Programmbeispiel zu diesem Thema nutzt ein ebenfalls per Buchdiskette mitgeliefertes MessageHook-Control.

Wenn man ohnehin nicht ohne Zusatz-Controls auskommt - warum dann nicht die gesamte COM-Funktionalität in ein benutzerdefiniertes Steuerelement (Custom Control) stecken? Diese Idee steht hinter MSCOMM.VBX und wurde genial in die Tat umgesetzt.

Das MSComm-Control

Lobenswerterweise ist - zumindest mit der VB Professional Edition - ein Custom Control namens **MSCOMM.VBX** enthalten, das Sie Ihnen die API-Programmierung vollständig abnimmt. Die Einstellung von COM-Parametern sowie das Lesen und Schreiben serieller Daten geschieht bequemerweise mit Eigenschaften und Ereignissen; den Aufruf der zur Durchführung der jeweiligen Operation erforderlichen Treiberfunktionen übernimmt MSComm intern. Dieses Control wurde für Microsoft vom Tool-Hersteller **Crescent Software** entwickelt.

Durch Organisation der Kommunikationsfunktionen als Steuerelement bekommt man die serielle Schnittstelle sozusagen "zum Anfassen": Ein MSComm-Control entspricht einem COM-Port; benutzt man beispielsweise zwei COM-Schnittstellen, sieht man eben zwei MSComm-Controls vor. Genau wie ein Custom Control verfügt ein COM-Port über *Eigenschaften* (z.B. die Einstellung der Übertragungsparameter und die logischen Pegel der Steuerleitungen) sowie über Ereignisse (z.B. das Eintreffen eines Zeichens). Daher eignet sich ein solches VBX ideal zur Verkörperung eines COM-Ports - man hat das Gefühl, durch Einstellen und Auswerten von Eigenschaften und Verarbeitung von Ereignissen direkt am COM-Port zu "basteln".

Haben Sie sich bisher an die Verwendung von MSComm.VBX noch nicht herangetraut? Die Handhabung ist extrem einfach! Zunächst sollen die grundlegenden Merkmale dieses Controls erläutert werden. Anschließend geht's dann gleich mit ein paar praktischen Beispielen weiter.

Anmerkung: Für Benutzer der VB 2.0/3.0 Standardausführung in deutsch oder englisch - aber natürlich ebenso für die Professional-Version - steht mit dem in der Toolbox PDQComm for Windows enthaltenen PDQComm.VBX ein zu MSComm.VBX aufwärtskompatibles und stark erweiterertes Kommunikations-Control zur Verfügung. Alles in diesem Beitrag Gesagte gilt daher auch für PDQComm.VBX.

Kurzer Exkurs ins DOS-BASIC...

Bevor wir mit dem MSComm-Control loslegen, lassen Sie uns einen kurzen Exkurs in die DOS-Basic-Ära unternehmen. Viele Leser dürften bereits Erfahrung mit Microsoft QuickBASIC, BASIC PDS, VB/DOS oder einfach dem mit DOS gelieferten QBASIC gesammelt haben. Wer schon mit einem dieser Basic-Dialekte Kommunikations-Anwendungen erstellt hat, kennt die Vorgehensweise: Sämtliche Schnittstellen-Parameter werden hier beim Öffnen der Schnittstelle im Befehl *OPEN COM* angegeben. Die Syntax dazu gestaltet sich wie folgt:

```
OPEN "COMn: Opt1 Opt2" [FOR Modus] AS [#]DateiNr% [LEN=Satzlänge%]
```

Eine typische *OPEN*-Anweisung zum Öffnen eines COM-Ports sieht unter DOS-BASIC-Dialekten daher etwa wie folgt aus:

```
OPEN "COM2:1200,N,8,1,ASC,CD0,CS0,DS0,RS,TB2048,RB2048" AS #1 LEN=100
```

Hier wird COM2 für 1200 Baud, keine Parität, und 8 Bit Wortbreite geöffnet; Sende- und Empfangspuffer wurden auf jeweils 2048 Bytes eingestellt. Hardware-Handshake wird unterdrückt.

Neben dem Lesen und Schreiben mit mit den üblichen Befehlen *Get*, *Put* und *Print* steht dem DOS-BASIC-Programmierer die Anweisung *ON COM GOSUB* ... zur Verfügung. Sie erlaubt es, beim Eintreffen von Zeichen zu einem bestimmten Programmabschnitt zu verzweigen.

COMn: ist der zu öffnende COM-Port (1 = COM1, 2 = COM2 usw.)

Mit *Opt1* werden die am häufigsten verwendeten Übertragungsparameter angegeben, nämlich:

[Baud] [,[Parität] [,[Daten] [,[Stop]]]]

Baud ist die Übertragungsrate (300, 600, 1200, 2400 etc.).

Parität gibt an, wie die Parität überprüft werden soll (N = keine Prüfung, E = gerade, O = ungerade).

Daten ist die Wortbreite (5, 6, 7 oder 8 Datenbits).

Stop ist die Anzahl der Stopbits (1, 1.5 oder 2). Diese Werte sind optional verwendbar und haben - wenn nichts angegeben wird - die Standardeinstellung 300 Baud, gerade Parität, 7 Datenbits und 1 Stopbit.

Mit *Opt2* werden seltener verwendete Parameter eingestellt, die durch durch Kommas getrennt sind. Unter anderem sind folgende Parameter spezifizierbar:

CD[n], CS[n] und *DS[n]* geben für die Steuerleitungen CD, CTS und DSR die Fehlerwartezeit (Timeout) in ms an.

LF sendet nach jedem Wagenrücklaufzeichen (CR) ein Zeilenvorschubzeichen (LF).

OP[m] legt fest, wie lange *OPEN COM* warten soll, bis alle Datenübertragungs-Verbindungen geöffnet sind (in ms).

RB[n] und *TB[n]* legen die Größe des Empfangs- bzw. Sendepuffers fest (in Byte).

RS unterdrückt das Erkennen einer Sendeanforderung. *ASC* bzw. *BIN* öffnet den Port im ASCII- bzw. Binär-Modus.

Modus legt - vergleichbar mit Diskdatei-Zugriff - den Dateimodus fest, und zwar INPUT, OUTPUT oder RANDOM (Standardeinstellung).

Mit *Dateinr%* wird eine Kanalnummer zugewiesen (1 bis 255), unter deren Angabe nach dem Öffnen auf den Port zugegriffen werden kann.

Mit *Satzlänge%* läßt sich noch die Puffergröße im Direktzugriffs-Modus einstellen; die Vorgabe beträgt 128 Bytes.

... Und zurück zu VB/Win

Wie sieht dies nun beim Einsatz von Visual Basic für Windows und dem Custom Control MSComm.VBX aus? Im Prinzip stehen hier dieselben Möglichkeiten zur Verfügung, nur eben VB auf den Leib geschneidert.

Beim DOS-BASIC geschieht die Einstellung aller Übertragungsparameter im OPEN-Befehl zum Öffnen der Schnittstelle. Bei MSComm stehen zum Einstellen der Übertragungsparameter eine Reihe von **Eigenschaften** zur Verfügung; sie haben ebenfalls bestimmte Grundeinstellungen. Wir wollen sie nachfolgend den entsprechenden DOS-BASIC-Parametern gegenüberstellen.

```
OPEN "COM2: 1200,N,8,1, ASC, CD0,CS0,DS0, RS, TB2048,RB2048" AS #1 LEN=100
```

Klicken Sie in der Syntax-Zeile einfach auf den jeweiligen Parameter, um die entsprechende Eigenschaft des MSComm-Controls anzuzeigen.

Das *Schließen* geschieht unter DOS-BASIC mit dem Befehl *CLOSE Kanalnr.* Bei MSComm wird stattdessen *PortOpen* auf falsch gesetzt (*PortOpen = False*).

Außerdem gibt es noch eine Reihe weiterer MSComm-Eigenschaften, zu denen es kein DOS-BASIC-Gegenstück gibt:

- Break
- ParityReplace
- Interval
- Notification
- DTREnable/RTSEnable
- RThreshold
- SThreshold
- InBufferCount
- OutBufferCount

Dem OPEN-Schlüsselwort selbst entspricht die MSComm-Eigenschaft *PortOpen*. Zum Öffnen wird sie auf wahr eingestellt (*PortOpen=True*).

In der Eigenschaft *CommPort* wird die Nummer des zu verwendenden COM-Ports festgelegt. Dies kann jede Nummer zwischen 1 und 99 sein; in der Regel dürfte sich die Angabe auf 1 - 4 (COM1 bis COM4) beschränken.

In *Settings* wird ein String mit den Einstellungen der Baudrate, Parität, Wortbreite und der Anzahl der Stopbits angegeben; die Schreibweise entspricht dem hervorgehobenen Bereich der DOS-BASIC-Zeile. Zulässige Baudraten sind die Standard-Baudraten von 110 bis 19200. Höhere Baudraten (bis 256000) sind in der Online-Hilfe als reserviert ausgewiesen; sie werden nur vom PDQComm-Control unterstützt.

Während der *OPEN COM*-Befehl von DOS-BASIC wahlweise das Öffnen im ASCII- oder Binär-Modus erlaubt, ist MSComm fest auf Binär-Modus eingestellt. Es werden stets sämtliche Zeichen gesendet bzw. empfangen. Das ist auch gut so, denn Funktionalität wie beispielsweise das Anhängen von Wagenrücklauf- und Zeilenvorschubzeichen läßt sich bei Bedarf leicht selbst bewerkstelligen. Um den Empfang eines Dateiende-Zeichens (EOF, ASCII 26) aufspüren zu können, wurde eine spezielle Konstante für das weiter hinten erläuterte *OnComm*-Ereignis vorgesehen. Mit der logischen Eigenschaft *NullDiscard* kann außerdem gewählt werden, ob am Port eintreffende Nullzeichen in den Empfangspuffer übernommen werden.

Es werden nicht nur sämtliche Zeichen gesendet; es werden auch keine hinzugefügt. Daher gibt es bei MSComm auch kein Äquivalent zum *OPEN COM*-Parameter LF (sendet nach einem Wagenrücklaufzeichen ein Zeilenvorschubzeichen). CR- bzw. CR/LF-Codes zum Zeilenabschluß müssen bei Bedarf selbst hinzugefügt werden.

Mit *CDTimeout*, *CTSTimeout* und *DSRTimeout* geben Sie für die Steuerleitungen CD, CTS und DSR die Fehlerwartezeit (Timeout) in ms an. Für die Logikpegel dieser Steuerleitungen sind die Eigenschaften *CDHolding*, *DSRHolding* und *CTSHolding* zuständig.

Beim DOS-BASIC unterdrückt die Angabe von RS das Erkennen einer Sendeanforderung (Request To Send, RTS). Bei MSComm erfolgt dies durch entsprechende Einstellung der Eigenschaft *Handshaking*.

Die Größe der Sende- und Empfangspuffer ist am Control mit den Eigenschaften *OutBufferSize* und *InBufferSize*einstellbar. Je größer Sie diese Puffer einstellen, um so weniger Speicher steht Ihnen anderweitig zur Verfügung. Andererseits bewirken zu kleine Puffer schneller einen Überlauf; besonders bei hohen Übertragungsraten kann es dann passieren, daß der ausführende Programmcode die Zeichen des Datenstroms nicht schnell genug verarbeiten kann. Der Sendepuffer sollte mindestens 512 Byte, der Empfangspuffer 1024 Byte groß sein (Grundeinstellung); im Zweifelsfalle sollte es lieber etwas mehr sein.

Eine Kanalnummer brauchen wir bei Verwendung des Controls nicht zuzuweisen. Vielmehr wird für jeden Port, den wir benutzen wollen, ein eigenes MSComm-Control vorgesehen; auf den jeweiligen Port bezieht man sich dann eben nicht per Kanalnummer, sondern durch Angabe des Control-Namens (z.B. *Comm1.Input*).

Mit der Eigenschaft *InputLen* wird festgelegt, wieviele Zeichen mit jeder Leseoperation aus dem Eingangspuffer gelesen werden; sie läßt sich daher in etwa mit der Satzlänge von *OPEN COM* vergleichen.

In vielen Punkten gehen die vom MSComm-Control gebotenen Möglichkeiten über DOS-BASIC hinaus. So unterstützt *OPEN COM* beispielsweise keinen Software-Handshake (XOn/Off). MSComm hingegen läßt Sie mit der Eigenschaft *Handshaking* wählen, ob Software-Handshake (1), Hardware-Handshake (2), beides (3) oder keines von beiden (0) erfolgen soll.

Diese Eigeschaft läßt sich zur Laufzeit auf *True* setzen, um ein Break-Signal (Pause) zu senden. Das Senden von Zeichen wird dann solange eingestellt, bis Break wieder auf *False* gesetzt wird. Typischerweise wird dies nur für kurze Zeiträume durchgeführt, und nur dann, wenn das Gerät am anderen Ende dies anfordert.

Diese Eigenschaft setzt oder liefert das Zeichen, mit dem bei Auftreten eines Paritätsfehlers ungültige Zeichen im Datenstrom ersetzt werden (*MSComm.ParityReplace[= Zeich$]*). Die Grundeinstellung sieht als Ersatzzeichen ein Fragezeichen (?) vor. Die Angabe eines Leerstrings ("") für *ParityReplace* schaltet die Paritätsprüfung aus.

Hier wird in Millisekunden eingestellt, in welchem Zeitabstand die Hardware abgefragt werden soll (*MSComm.Interval[= milliseconds&]*). Unter Windows 3.0 war diese als *Polling* bezeichnete Methode die einzig verfügbare. Die Grundeinstellung beträgt bei der aktuellen MSComm-Version 55 ms (früher waren es 1000 ms).

Ab Windows 3.1 ist noch eine weitere Methode zur COM-Port- Abfrage hinzugekommen, die als "Comm Notification" bezeichnet wird. Bei diesem Verfahren schickt Windows bei Eintreffen eines jeden Zeichens eine Benachrichtigung an MSComm. Als zuverlässiger hat sich jedoch das Polling-Verfahren von Windows 3.0 erwiesen, das auch bei neueren Windows-Versionen nach wie vor zur Verfügung steht. Mit der in der aktuellen MSComm-Version neu hinzugekommenen und nur zur Laufzeit verfügbaren Eigenschaft *Notification* wird bestimmt, ob das Polling-Verfahren verwendet wird (*Notification* = *False*) oder Comm Notification (*Notification* = *True*). Letzteres empfiehlt Microsoft nur bei der Verwendung von Modems mit weniger als 14400 bps.

Durch Angabe von wahr oder falsch kann mit diesen Eigenschaften festgelegt werden, ob die Steuerleitungen Data Terminal Ready (DTR) bzw. Request To Send (RTS) freigeschaltet werden sollen. In diesem Fall wird die jeweilige Eigeschaft auf *True* gesetzt, andernfalls auf *False*. Bei *True* hat die jeweilige Leitung nach dem Öffnen des Ports High-Pegel, bei geschlossenem Port Low-Pegel. Bei *False* hat die jeweilige Leitung stets Low-Pegel.

Mit dieser Eigenschaft wird mit einem Integerwert angegeben, nach welcher Anzahl empfangener Zeichen die Eigenschaft *CommEvent* auf den Wert *MSCOMM_EV_RECEIVE* gesetzt und das Ereignis *OnComm* ausgelöst wird (*MSComm.RThreshold[= AnzZeich%]*). Wird beispielsweise *RThreshold* gleich 1 gesetzt, löst das Control bei jedem einzelnen Zeichen, das in den Empfangspuffer wandert, das *OnComm*-Ereignis aus. Durch Angabe von null (0) läßt sich das *OnComm*-Ereignis beim Zeichenempfang unterdrücken; dies ist zugleich die Grundeinstellung von MSComm.

Dies ist die entsprechende Eigenschaft für den Sendepuffer; auch hier wird die Anzahl der Zeichen per Integerwert angegeben (*MSComm.SThreshold[= AnzZeich%]*). Sinkt die Anzahl der Zeichen im Sendepuffer unter *AnZeich%* , wird die Eigenschaft *CommEvent* auf den Wert *MSCOMM_EV_ SEND* gesetzt und das Ereignis *OnComm* ausgelöst. Bei *MSComm.SThreshold = 1* erzeugt das Control das OnComm-Ereignis, sobald der Sendepuffer völlig geleert ist. Das Sende-*OnComm*-Ereignis wird nur einmal - bei Unterschreiten des eingestellten Schwellenwertes - ausgelöst. Bei Angabe von null (0) wird die Auslösung eines Sende-OnComm-Ereignisses unterdrückt (Grundeinstellung).

Diese Eigenschaft liefert als Integerwert die Anzahl der im Empfangspuffer wartenden Zeichen, die Sie von dort mit der *Input*-Eigenschaft holen können. Durch Angabe von *InBufferCount = 0* können Sie den Empfangspuffer löschen, ohne ihn auszulesen. *InBufferCount* ist nur zur Laufzeit verfügbar und sollte nicht mit *InBufferSize* (Größe des eingerichteten Empfangspuffers) verwechselt werden.

Diese Eigenschaft liefert als Integerwert die Anzahl der im Sendepuffer wartenden Zeichen. Bei Bedarf können Sie den Sendepuffer durch Angabe von *OutBufferCount = 0* löschen, statt ihn mit der *Output*-Eigenschaft zu senden. *OutBufferCount* ist nur zur Laufzeit verfügbar und sollte nicht mit *OutBufferSize* (Größe des eingerichteten Sendepuffers) verwechselt werden.

COM-Ereignisse

Anstelle des *ON COM GOSUB* verfügt MSComm - ganz im Sinne des VB/Win-Grundgedankens - über ein *Ereignis*; analog zum DOS-BASIC heißt es *OnComm.* Bei Auftreten dieses Ereignisses gibt die Eigenschaft *CommEvent* darüber Aufschluß, was genau passiert ist - je nach geliefertem Zahlencode ein bestimmtes Kommunikationsereignis (zum Beispiel eintreffende Zeichen) oder ein bestimmter Fehler. Die *Tabelle* zeigt Ihnen alle *CommEvent*-Konstanten im Überblick.

Ein-/Ausgabe

Zum Lesen und Schreiben von Daten über die COM-Schnittstelle dienen die Control-Eigenschaften *Input* und *Output*. Allgemein fällt auf, daß die MSComm-Entwickler auch zur Ausführung von Aktionen mehr von Eigenschaften als von Methoden Gebrauch gemacht haben.

Jetzt aber endlich ein Beispiel!

Genug der Theorie - Nachdem Sie nun sämtliche Eigenschaften und Ereignisse des MSComm-Controls kennengelernt haben, soll ein praxisnahes Beispiel folgen.

1. Beginnen Sie zum Ausprobieren ein neues Projekt mit einer leeren Form.

2. Laden Sie MSComm.VBX hinzu (in der Regel wurde es im Verzeichnis \WINDOWS\SYSTEM installiert) und ziehen Sie das Control auf Ihre Form.

3. Gehen Sie dann ins Code-Fenster der *Form_Load*-Prozedur. Sämtlicher Programmcode dieses kleinen Beisiel ist dort einzugeben.

Um mit einem COM-Port Daten lesen bzw. schreiben zu können, müssen zunächst die Nummer des zu verwendenden COM-Ports und dessen Übertragungsparameter eingestellt werden. Für unseren Test verwenden wir die Schnittstelle COM2, da COM1 in der Regel mit der seriellen Maus belegt ist; Sie können jedoch auch einen anderen in Ihrem System installierten COM-Port angeben. Als Übertragungsparameter verwenden wir 1200 Baud, keine Parität, 8 Datenbits und 1 Stopbit. *InputLen* setzen wir auf null, um mit *Input* den kompletten Puffer lesen zu können:

```
'COM2 verwenden:
Comm1.CommPort = 2
Comm1.Settings = "1200,N,8,1"
Comm1.InputLen = 0
```

Anschließend muß die serielle Schnittstelle für die Dauer der Benutzung geöffnet werden:

```
'COM-Port öffnen:Comm1.PortOpen = True
```

Damit wir den spannenden Moment der Übertragung nicht verpassen, sehen wir eine Messagebox mit OK-Button vor, bei dessen Betätigung das Senden der Testdaten gestartet wird:

```
MsgBox Ein$ + "Fertig zum Senden zu COM" + Str$(Comm1.CommPort)
```

Verwenden Sie ein (Hayes-kompatibles) Modem, verwenden Sie folgende Zeile zum Senden; sie schickt einfach den Befehl "AT" <Eingabetaste>, was sich das Modem durch Zurücksenden von "OK" bestätigt:

```
Comm1.Output = "AT" + Chr$(13)
```

Falls Sie zum Testen zwei Computer per Nullmodem-Kabel verbunden haben, starten Sie am anderen Gerät das Windows-Terminalprogramm und stellen Sie dort dieselben Übertragungsparameter ein. In unserem Beispiel geben Sie stattdessen folgende Zeile ein; sie sendet testhalber einfach die Uhrzeit zum anderen PC:

```
Comm1.Output = Time$ + Chr$(13)
```

Nach dem Senden warten wir in einer Leerschleife, bis im Empfangspuffer zwei Zeichen eingetroffen sind - beim Modem das "OK", bei der PC-PC-Kopplung sind am anderen Gerät zwei Zeichen im Terminalfenster einzutippen. Ganz leer ist die Schleife übrigens nicht - durch Einfügen der *DoEvents*-Anweisung sorgen wir dafür, daß anliegende Ereignisse anderer Anwendungen verarbeitet werden können:

```
Do
    DoEvents
Loop Until Comm1.InBufferCount >= 2
```

Nachdem zwei Zeichen empfangen wurden, lesen wir diese aus dem Empfangspuffer:

```
Ein$ = Comm1.Input
```

Damit ein COM-Port bei Bedarf auch anderen Anwendungen zur Verfügung steht, sollten Sie ihn nur solange wie nötig geöffnet halten. Darum wollen wir ihn gleich wieder schließen:

```
Comm1.PortOpen = False
```

Aussschließlich zu unserer Information zeigen wir in einer Messagebox, daß das Programm beendet wurde; anschließend verlassen wir es:

```
'Mitteilung zur Kontrolle
MsgBox Ein$ + " empfangen. Port geschlossen."
End
```

Hat´s geklappt? Falls nicht, überprüfen Sie die verwendeten Übertragungsparameter, den gewählten COM-Port und schauen gegebenenfalls nach, ob das Modem bzw. Der andere PC betriebsbereit und ordnungsgemäß verbunden ist.

Wie bereits erwähnt, überträgt MSComm Daten stets binär; es wird nichts weggefiltert oder hinzugefügt. Manuell an ein Modem gesendete AT-Befehle müssen durch Drücken der Eingabetaste abgeschlossen werden. Entsprechend muß beim Senden eines solchen Befehls per Programm ein Wagenrücklauf-Zeichen hinterhergeschickt werden, daher *Comm1.Output = "AT" + Chr$(13).* Soll auch noch ein Zeilenvorschub erfolgen, ist auch dessen Code hinzuzufügen, z.B. *A$+Chr$(13)+Chr$(10)*.

Falls Sie sich das Eintippen ersparen wollen: Auf der Basic-Professionell-Heftdiskette zu Ausgabe 3/94 finden Sie das obige Beispiel als Projekt VBCOMM1.MAK; des weiteren finden Sie es in Listing 1 "am Stück". Die folgenden beiden Beispiele lassen sich besonders gut ausprobieren, wenn man zwei Computer zur Verfügung hat, die per Nullmodem-Kabel gekoppelt werden (verschrotten Sie Ihren veralteten 286er Laptop-Rechner nicht - er eignet sich immer noch hervorragend als COM-Tester!).

```
'Listing 1 - Ein erstes Beispiel
'Erläuterungen siehe Text, zu dem dieses Listing gehört

Sub Form_Load ()
   Form1.Show
   Comm1.CommPort = 2                  'COM2 verwenden (MUSS vorm Öffnen
angegeben werden)
   Comm1.Settings = "1200,N,8,1"    '1200 Baud, keine Parität, 8 Datenbits, 1
Stopbit
   Comm1.InputLen = 0                  'Mit Input den kompletten Puffer lesen.
   Comm1.PortOpen = True                           'COM-Port öffnen

   MsgBox Ein$ + "Fertig zum Senden zu COM" + Str$(Comm1.CommPort)


   Comm1.Output = Time$ + Chr$(13) + Chr$(10)'Für Modem: AT-Befehl senden
   'Comm1.Output = Time$ + Chr$(13)+chr$(10   'Für PC/PC-Verbindung: Uhrzeit
ausgeben

   Do                                      'Auf serielle Empfangsdaten warten
      DoEvents
   Loop Until Comm1.InBufferCount >= 2   'Mehr als 2 Zeichen im Puffer -->
Modem
                                  'hat mit "OK" geantwortet
                                  'PC/PC: Am anderen PC "OK" tippen
   Ein$ = Comm1.Input                   'COM-Port lesen/leeren

   Comm1.PortOpen = False              'COM-Port schließen

                           'Mitteilung zur Kontrolle
   MsgBox Ein$ + " empfangen. Port geschlossen."
   End

End Sub
```

Ein einfaches Terminal-Programm mit Do Loop ....

Als nächstes Beispiel wollen wir ein ganz einfaches Terminal-Programm erstellen, an das wir noch keine besonderen Ansprüche stellen wollen; es soll lediglich das Prinzip gezeigt werden. Um die flexiblen Einsatzmöglichkeiten des MSComm-Controls zu demonstrieren, wird das Programm auf zweierlei Weise realisiert. Im ersten Ansatz des Programms benutzen wir - wie auch schon eben in unserem kleinen Einführungsbeispiel - zum Empfang der Daten eine Schleife. Wie Sie in *Listing 2* sehen, wird das ganze auch hier in der *Form_Load*-Prozedur untergebracht.

Als Terminal-Fenster verwenden wir ein VB-Textfeld, dessen *Multiline*- und *Scrollbar*-Eigenschaften wir eingeschaltet haben.

Mit Kenntnis des Einführungsbeispiels ist der Inhalt der *Do-Loop*-Schleife ist eigenlich selbsterklärend. Empfangene Zeichen fügen wir der *Text*-Eigenschaft des Textfeldes hinzu. Im *Select-Case*-Block wurde eine Prüfung vorgesehen, um welche Codes es sich bei den empfangenen Zeichen handelt. Im Falle eines Wagenrücklaufzeichens fügen *Text1.Text* noch ein Zeilenvorschub-Zeichen hinzu, um eine neue Zeile zu beginnen. Die Positionierung des Textcursors jeweils hinter das letzte Zeichen des Textfeldes geschieht mit den Anweisungen *SelText* und *SelLength*. Demonstrationshalber wird auch das EOF-Zeichen (Dateiende) ausgewertet. Den Füllstand *InBufferCount* des Empfangspuffers zeigen wir zur Information in der Titelleiste der Form an.

Um mit unserem "Arme-Leute-Terminalprogramm" auch senden zu können, werten wir in der Prozedur *Text1_KeyPress* die Tastendrücke aus und senden deren Ansi-Codes zum COM-Port.

Das Schließen des COM-Ports geschieht in der *Unload*-Prozedur der Form.

Auch dieses VB-Projekt brauchen Sie nicht einzutippen; Sie können es Prozedur für PRozedur aus dieser HLP-Datei kopieren und Sie finden es auf der Basic-Professionell-Heftdiskette unter dem Namen VBCOMM2.MAK.

```
'Listing 2 - Einfaches Terminal-Programm mit Do Loop
'Erläuterungen siehe Text, zu dem dieses Listing gehört

Sub Form_Load ()
   Comm1.CommPort = 2                  'COM2 verwenden (VOR dem Öffnen
angeben!)
   Comm1.Settings = "1200,N,8,1"   '1200 Baud, keine Parität, 8 Datenbits, 1
Stopbit
   Comm1.InputLen = 1                   'Mit Input je 1 Zeichen lesen
   Comm1.OutBufferSize = 512          'Sendepuffer
   Comm1.InBufferSize = 1024          'Empfangspuffer
   Comm1.PortOpen = True               'COM-Port öffnen
   Form1.Show

   Do
       DoEvents
       If Comm1.InBufferCount > 0 Then
       Form1.Caption = "Im Empfangspuffer: " + Str$(Comm1.InBufferCount)
       Ein$ = Comm1.Input
       Select Case Asc(Ein$)
       Case 26    'hier kann auch Dateiende (EOF) abgefragt werden
           Comm1.PortOpen = False
           MsgBox "Dateiende", 16, "Übertragung"
           End
       Case 13    'CR
           Ein$ = Ein$ + Chr$(10)
           Text1.Text = Text1.Text + Ein$
       Case Else
           Text1.Text = Text1.Text + Ein$
       End Select
       Text1.SelStart = Len(Text1.Text)
       Text1.SelText = ""
       End If
   Loop

End Sub

Sub Form_Unload (Cancel As Integer)
   Comm1.PortOpen = False             'COM-Port schließen
End Sub

Sub Text1_KeyPress (KeyAnsi As Integer)
   Comm1.Output = Chr$(KeyAnsi)
End Sub
```

... und mit OnComm-Ereignis

Die im vorangegangenen Programmbeispiel gezeigte Methode funktioniert zwar, hat aber einen Nachteil: Durch Verzicht auf die Benutzung von *OnComm* bleiben uns die von diesem Ereignis gelieferten Informationen versagt. Wie bereits erläutert, steht bei Auftreten des *OnComm*-Ereignisses in der Eigenschaft *CommEvent*, was genau passiert ist - je nach geliefertem Zahlencode ein bestimmtes Kommunikationsereignis oder ein Fehler.

Darum wurde für die in *Listing 3* gezeigte Programm-Variante ein anderer Ansatz gewählt - die Nutzung des *OnComm*-Ereignisses. Das Listing können Sie aus der HLP-Datei Prozedur für Prozedur kopieren.

Das Projekt wurde um ein Basic-Modul mit den Definitionen der Ereignis- und Fehler-Konstanten des MSComm-Controls ergänzt, wie Sie sie auch inKommunikationsereignis- und Fehlertabellen (siehe oben) sehen. Auf der Basic-Professionell-Heftdiskette heißt diese Datei MSCOMM.BAS; die Definitionen sind jedoch auch in der mit VB 3.0 Professional ausgelieferten Datei CONSTANT.TXT enthalten (das komplette Projekt finden Sie auf der aktuellen Heftdiskette als VBCOMM3.MAK).

Die *OnComm*-Ereignisprozedur wird abgearbeitet, wann immer eines der in *Tabelle 4* aufgeführten Situationen eintritt. Um zu unterscheiden, um welche Situation es sich jeweils handelt, sehen wir einen *Select-Case*-Block mit den von MSComm unterstützten Ereignis- und Fehler-Konstanten vor.

Für die Anzeige eventuell auftretender Fehler wurde als Statusanzeige ein Label-Control *lblStatus* ergänzt; je nach Fehlercode zeigen wir hier eine Textmitteilung an. Auch für einige Ereignisse wie Pegeländerungen der Steuerleitungen wurden Meldungen vorgesehen.

Im Falle empfangener Zeichen (*Case MSCOMM_EV_RECEIVE*) fügen wir diese dem Textfeld hinzu. Da wir *Comm1.RThreshold = 1* angegeben haben, wird bei jedem empfangenen Zeichen ein Ereignis ausgelöst. Für geschwindigkeitskritische Anwendungen ist es oft vorteilhafter, zu "sammeln", bevor man ein Ereignis auslösen läßt und Programmcode ausführt; hierzu wird für *Comm1.RThreshold* ein entsprechend höherer Wert angegeben. Die Textcursor-Positionierung geschieht wie beim vorangegangenen Beispiel.

```
'Listing 3 -   Einfaches Terminal-Programm mit OnComm
'Erläuterungen siehe Text, zu dem dieses Listing gehört

Sub Comm1_OnComm ()
Static Zeile$
lblStatus.Caption = ""             'Statuszeile löschen (optional)

If Comm1.CommEvent > 1000 Then   'Rote Statusmeldung für Fehler ...
    lblStatus.ForeColor = &HFF&
Else
    lblStatus.ForeColor = &H0&    '... sonst schwarzer Text
End If

Select Case Comm1.CommEvent
'Ereignisse
    Case MSCOMM_EV_CD
        lblStatus.Caption = " Pegeländerung der CD-Leitung"
    Case MSCOMM_EV_CTS
        lblStatus.Caption = " Pegeländerung der CTS-Leitung"
    Case MSCOMM_EV_DSR
        lblStatus.Caption = " Pegeländerung der DSR-Leitung "
    Case MSCOMM_EV_RING
        lblStatus.Caption = " RING (Läuten)"
    'EMPFANG
    Case MSCOMM_EV_RECEIVE   'RThreshold Anzahl Zeichen empfangen
        Form1.Caption = "Im Empfangspuffer: " + Str$(Comm1.InBufferCount)
        Ein$ = Comm1.Input
        If Len(Ein$) > 0 Then
       If Asc(Ein$) = 13 Then
           lblStatus.Caption = "CR empfangen"
           Ein$ = Ein$ + Chr$(13) + Chr$(10) 'Neue Zeile in Text1 erzwingen
(optional)
       End If
        End If
        Text1.Text = Text1.Text + Ein$
        Text1.SelStart = Len(Text1.Text)
        Text1.SelText = ""

    Case MSCOMM_EV_SEND
        lblStatus.Caption = " In SThreshold angegebene Zeichenanzahl
unterschritten"

    Case MSCOMM_EV_EOF
        lblStatus.Caption = "Dateiende-Zeichen (EOF) empfangen"

'Fehlercodes (> 1000)
    Case MSCOMM_ER_BREAK
        lblStatus.Caption = " Break-Signal empfangen"
        'hier ggf. Code zur Break-Behandlung vorsehen...
    Case MSCOMM_ER_CDTO
        lblStatus.Caption = " Carrier Detect Timeout"
    Case MSCOMM_ER_CTSTO
        lblStatus.Caption = " Clear To Send Timeout"
    Case MSCOMM_ER_DSRTO
        lblStatus.Caption = " Data Set Ready Timeout"
    Case MSCOMM_ER_FRAME
        lblStatus.Caption = " Fehlerhafte Rahmen-Bits"
```

```
    Case MSCOMM_ER_OVERRUN
        lblStatus.Caption = " Overrun (Zeichen 'verschluckt')"
    Case MSCOMM_ER_RXOVER
        lblStatus.Caption = " Empfangspuffer-Überlauf"
    Case MSCOMM_ER_RXPARITY
        lblStatus.Caption = " Paritätsfehler (Empfang)"
    Case MSCOMM_ER_TXFULL
        lblStatus.Caption = " Sendepuffer voll"
End Select

End Sub

Sub Form_Load ()
   Comm1.CommPort = 2                  'COM2 verwenden (VOR dem Öffnen
angeben!)
   Comm1.Settings = "1200,N,8,1"    '1200 Baud, keine Parität, 8 Datenbits, 1
Stopbit
   Comm1.InputLen = 1                   'Mit Input je 1 Zeichen lesen
   Comm1.OutBufferSize = 512          'Sendepuffer
   Comm1.InBufferSize = 1024          'Empfangspuffer (testweise =1 setzen, um
                          'Überlauffehler zu provozieren)
   Comm1.RThreshold = 1                'Füllstand bis zum OnComm-Ereignis
                          '(hier: Ereignis jedem Zeichen)
   Comm1.PortOpen = True               'COM-Port öffnen

End Sub

Sub Form_Unload (Cancel As Integer)
   Comm1.PortOpen = False             'COM-Port schließen
End Sub

Sub Text1_KeyPress (KeyAnsi As Integer)
   Comm1.Output = Chr$(KeyAnsi)
End Sub
```

```
'--------------------------------------
'Konstanten für MSComm-Control
'--------------------------------------
'Quittungsbetrieb
Global Const MSCOMM_HANDSHAKE_NONE = 0
Global Const MSCOMM_HANDSHAKE_XONXOFF = 1
Global Const MSCOMM_HANDSHAKE_RTS = 2
Global Const MSCOMM_HANDSHAKE_RTSXONXOFF = 3

'CommEvent-Ereignisse
Global Const MSCOMM_EV_SEND = 1
Global Const MSCOMM_EV_RECEIVE = 2
Global Const MSCOMM_EV_CTS = 3
Global Const MSCOMM_EV_DSR = 4
Global Const MSCOMM_EV_CD = 5
Global Const MSCOMM_EV_RING = 6
Global Const MSCOMM_EV_EOF = 7

'CommEvent-Fehlercodes
Global Const MSCOMM_ER_BREAK = 1001
Global Const MSCOMM_ER_CTSTO = 1002
Global Const MSCOMM_ER_DSRTO = 1003
Global Const MSCOMM_ER_FRAME = 1004
Global Const MSCOMM_ER_OVERRUN = 1006
Global Const MSCOMM_ER_CDTO = 1007
Global Const MSCOMM_ER_RXOVER = 1008
Global Const MSCOMM_ER_RXPARITY = 1009
Global Const MSCOMM_ER_TXFULL = 1010
```

Fragen & Antworten zu MSComm

F: Wie stelle ich sicher, daß der zum Öffnen angegebene COM-Port auf dem Gerät auch verfügbar ist?

A: Erkennt das Control beim Öffnen mit *PortOpen*, daß der angegebene COM-Port im verwendeten PC nicht vorhanden ist, erzeugt es den Fehlercode 68 ("Gerät nicht verfügbar").

```
'COM2 verwenden ...
Comm1.CommPort = 2
'Öffnen ...
Comm1.PortOpen = True
If Err Then
    MsgBox "COM2: nicht verfügbar - bitte anderen Port wählen", 16
    Exit Sub    'oder was auch immer
End If
```

F: Wann schließt man einen COM-Port am besten?

A: Das Schließen eines COM-Ports kann in der *Unload*-Prozedur z.B. der Hauptform geschehen; so wird der Port spätestens bei Verlassen des Programms geschlossen. Um zu prüfen, ob der Port noch geöffnet ist, können Sie seinen Status einfach abfragen:

```
If MSComm1.PortOpen Then ...
```

Gegebenenfalls empfiehlt es sich vor dem Schließen, durch Prüfen von *InBufferCount* und *OutBufferCount* sicherzustellen, daß alle empfangenen Daten aus dem Empfangspuffer gelesen und alle im Sendepuffer anstehenden Daten gesendet wurden.

F: Warum wurden in den Programmbeispielen Einstellungen wie *Comm1.Settings* im Programmcode definiert und nicht in der Control-Eigenschaftenliste eingestellt?

A: Es empfiehlt sich, alle für die serielle Übertragung wichtigen Eigenschaften in der *Form_Load*-Prozedur zu einzustellen oder einen Konfigurations-Dialog vorzusehen. Insbesondere auf Grundeinstellungen sollte man sich besser nicht verlassen - sie werden womöglich in aktualisierten Control-Versionen geändert, wie bei MSComm.VBX mit der *Interval*-Eigenschaft geschehen (jetzt 55 ms statt 1000 ms in der Vorgängerversion).

F: Wie empfange ich eine Datei?

A: Zum Beispiel so: Öffnen Sie zunächst eine Datei, in die Sie die zu empfangenden Zeichen hineinschreiben. In einer Do-Loop-Schleife prüfen Sie durch Abfrage von *InBufferCount*, ob Zeichen eintreffen. Ist dies der Fall, lesen Sie diese mit Input. Jetzt können Sie noch eine Filterung vornehmen. In unserem Beispiel prüfen wir zum Beispiel auf Dateiende-Zeichen (EOF) , um die Datei bei Empfang eines solchen Zeichens zu schließen; empfangene Wagenrücklaufzeichen (CR) ergänzen wir um Zeilenvorschub-Codes (LF). Beachten Sie jedoch, daß diese Art der Dateiübertragung keinerlei Fehlererkennung oder gar -Korrektur bietet. Zur Übertragung mit fehlersicheren Übertragungsprotokollen sollte man auf das erweiterte Kommunikations-Control in PDQComm for Windows zurückgreifen.

```
Sub Form_Load ()
   Comm1.CommPort = 2                  'COM2 verwenden (VOR Öffnen angeben!)
   Comm1.Settings = "1200,N,8,1"    '1200 Baud, keine Parität, 8 Datenbits, 1
```

```
Stopbit
   Comm1.InputLen = 1                  'Mit Input je 1 Zeichen lesen
   Comm1.OutBufferSize = 512          'Sendepuffer
   Comm1.InBufferSize = 1024          'Empfangspuffer
   Comm1.PortOpen = True               'COM-Port öffnen
   Form1.Show

   FileNum% = FreeFile      ' Nächste Dateinummer
   Open "Download.txt" For Binary As FileNum%

   Do
       DoEvents
       If Comm1.InBufferCount > 0 Then
           Form1.Caption = "Im Empfangspuffer: " + Str$(Comm1.InBufferCount)
           Ein$ = Comm1.Input
           Select Case Asc(Ein$)
           Case 26    'hier kann auch Dateiende (EOF) abgefragt werden
               'Close 1
               Comm1.PortOpen = False
               MsgBox "Dateiende", 16, "Download"
               Close FileNum%
               End
           Case 13    'CR
               Ein$ = Ein$ + Chr$(10)
               Put FileNum%, , Ein$
           Case Else
               Put FileNum%, , Ein$
           End Select
       End If
   Loop

End Sub
```

Fazit

Mit **MSComm** ist es Microsoft gelungen, Visual Basic um maßgeschneiderte Kommunikationsmöglichkeiten zu erweitern. Die Erklärungen und Programmbeispiele dieses sollten ausreichen, Ihnen bei der Kommunikation mit Visual Basic und dem MSComm-Control "aufs Fahrad zu helfen"; ein komplexeres Programmbeispiel - ein komplettes kleines Terminalprogramm - finden Sie in der Visual Basic Professional Edition selbst. Wer weiterfühende Kommunikationsmöglichkeiten benötigt, ist mit PDQComm for Windows gut beraten, denn er kann auf die mit MSComm gesammelten Erfahrungen aufbauen.

Autor dieses Online-Artikels ist **Dipl.-Ing. Harald Zoschke**. Herr Zoschke ist Geschäftsführer der **ZOSCHKE DATA GmbH**, die sich schwerpunktmäßig mit Entwicklung und Vertrieb von Visual-Basic-Tools sowie Werkzeugen zur Windows-Hilfeentwicklung beschäftigt. Sie erreichen Herrn Zoschke auf **CompuServe** unter **71340,2051**.

Literaturhinweise

[1] PDQComm for Windows, Handbuch, Crescent Software/Zoschke Data GmbH, Bahnhofstr. 3, D-24217 Schönberg

[2] Visual Basic Programmer´s Guide to the Windows API, Dan Appleman, Ziff-Davis Press, ISBN 1-56276-073-4, Bookshop VW02

[3] Visual Basic How-To, 2nd Ed., Robert Arnson, Waite Group Press, ISBN 1-878739-42-5, Bookshop VW13

Außerdem wird zum Thema Kommunikation und Connectivity in der Basic Professionell berichtet.

Bessere Kommunikationsunterstützung für Windows for Workgroups 3.11

Wenn Sie mit Microsoft Windows for Workgroups 3.11 arbeiten, können bei der Benutzung von Kommunikations-Software diverse Probleme auftreten, zum Beipiel "Aufhängen" bei Pentium-PCs mit UART-Chip vom Typ 16550 beim Versuch des Port-Öffnens, während sich Daten im Chip befinden. Auch bei anderen PCs kann es passieren, daß Sie mit Ihrer Kommunikations-Software nach dem Starten von WfWg 3.11 eine Verbindung herstellen können; bei weiteren Verbindungsversuchen hängt sich das System jedoch sogar auf. Zu diesen Problemen kommt es, wenn die zum WfWg-3.11-Kommunikationstreiber gehörige Datei SERIAL.386 beim Aufruf der API-Funktion CloseComm zusätzlich ein NUL-Zeichen zum Port schickt. Während dies normalerweise kein Problem darstellt, kann es im Falle weiterer Verbindungsversuche bei bestimmten Systemen zu Fehlfunktionen kommen. Zur Behebung bietet Microsoft auf Disk und per CompuServe-Forum eine verbesserte Version von SERIAL.386 an. Die alte Datei - sie steht in der Regel im Windows-System-Verzeichnis - benennt man sicherheitshalber um und kopiert dann die neue ins Verzeichnis, z.B. copy a:\serial.386 c:\windows\system. Leser der Basic Professionell finden SERIAL.386 nebst erklärender Textdatei auch auf der Heftdiskette zu Ausgabe 3/94.

TURBOCOM.DRV - Der COMM.DRV-Ersatz

Standardmäßig wird mit Windows 3.1x die Treiberdatei COMM.DRV geliefert; es gibt jedoch auch leistungsfähigere Alternativen. Um eine solche handelt es sich bei TURBOCOM.DRV. Das Produkt heißt Turbocom/2 und stammt von *Pacific CommWare*, 180 Beacon Hill Lane, Ashland, Oregon, 97520-9701. Telefon 503-482-2744, Fax 503-482-2627, BBS 503-482-2633. Ein solcher Treiber ist dringend erforderlich, wenn mit höheren Übertragungsraten gearbeitet werden soll; COMM.DRV ist auf 19200 beschränkt. Auf eine öffentliche CompuServe-Frage nach Erfahrungen antwortete ein Turbocom-Benutzer: "I am happy with TurboComm under Windows 3.1. I was having receive troubles with my laptop, but after Turbocomm installation - no problems and increased throughput."

MSCOMM.VBX-Update

MSCOMM.VBX ist in der VB/Pro-Ausführung seit Version 2.0 enthalten. Später wurde noch eine Reihe von Verbesserungen vorgenommen, die in die mit VB 3.0 Professional ausgelieferte MSComm-Version einflossen. Das aktualisierte Control ist jedoch auch auf CompuServe verfügbar, und zwar im amerikanischen Microsoft-Forum MSBASIC. Sie finden es dort als selbstextrahierende gepackte Datei mit dem Namen MSCOMM.EXE. Leser der Basic Professionell finden das aktuelle MSComm-Control auf der Heftdiskette zur Ausgebe 3/94 (zur Verwendung in der Entwicklungsumgebung ist *Visual Basic 3.0 Professional* erforderlich).

Kenndaten des ursprünglich mit Visual Basic 3.0 ausgelieferten MSCOMM.VBX

Datum: 28.4.1993
Zeit: 12:00 a.m.
Größe: 34304 Bytes
Version: 2.0.9000.7

Kenndaten des verbesserten MSCOMM.VBX:

Datum: 12.5.1993
Zeit: 12:21 p.m.
Größe: 34816 Bytes
Version: 2.1.0.1

Signal	9-polig	25-polig
CD (Carrier Detect)	1	8
RD (Empfang)	2	3
TD (Senden)	3	2
DTR (Data Terminal Ready)	4	20
GND (Masse)	5	7
DSR (Data Set Ready)	6	6
RTS (Request To Send)	7	4
CTS (Clear To Send)	8	5
RI (Ring Indicator)	9	22

Anschlußbelegung

Standard	Beschreibung		Baud	BPS
Bell 103	In den meisten 300-BPS-Modems		300	300
V.22	Hayes Smartmodem 1200mb		600	1200
V22bis	2400-BPS-Erweiterung für V.22		600	2400
V.32	9600 BPS Vollduplex	2400	9600	
V.32bis	14400-BPS-Erweiterung für V.32		2400	14400
V.42	Dito mit Fehlerkorrektur-Protokoll	-	-	
V.42bis	Lempel-Ziv-Datenkompression		-	38400

Wichtige DFÜ-Standards

Analoge VB-Dateioperation	Kommunikations-API-Funktion	Aufgabe
Open	OpenComm	COM-Port öffnen
	BuildCommDCB	COM-Parameter definieren
	SetCommState	COM-Parameter einstellen
Close	CloseComm	COM-Port schließen
Get	ReadComm	COM-Daten lesen
Put	WriteComm	COM-Daten schreiben
Err	GetCommError	Auf COM-Fehler prüfen

VB- und API-Funktionen im Vergleich

MSCOMM_EV_SEND	1	Im Sendepuffer wurde die in SThreshold angegebene Anzahl Zeichen unterschritten.
MSCOMM_EV_RECEIVE	2	Es wurde die in RThreshold angegebene Anzahl Zeichen empfangen. Dieses Ereignis wird kontinuierlich erzeugt, bis Sie die Daten mit Input aus dem Puffer entfernt haben.
MSCOMM_EV_CTS	3	Pegeländerung der CTS-Leitung (Clear To Send).
MSCOMM_EV_DSR	4	Pegeländerung der DSR-Leitung (Data Set Ready).
MSCOMM_EV_CD	5	Pegeländerung der CD-Leitung (Carrier Detect).
MSCOMM_EV_RING	6	Läuten (Ring) festgestellt (Achtung - dieses Ereignis wird von einigen UARTs nicht unterstützt).
MSCOMM_EV_EOF	7	Dateiende-Zeichen (EOF, ASCII 26) empfangen.

OnComm-Ereignis-Konstanten

MSCOMM_ER_BREAK	1001	Break-Signal empfangen.
MSCOMM_ER_CTSTO	1002	Clear To Send Timeout beim Versuch, ein Zeichen zu senden (siehe Eigenschaft CTSTimeout).
MSCOMM_ER_DSRTO	1003	Data Set Ready Timeout beim Versuch, ein Zeichen zu senden (siehe Eigenschaft DSRTimeout).
SCOMM_ER_FRAME	1004	Framing Error (fehlerhafte Rahmen-Bits im übertragenen Wort).
MSCOMM_ER_OVERRUN	1006	Overrun ("Verschlucken"; Zeichen noch nicht vom Port gelesen, bevor das nächste eintraf). Ggf. Interval ändern.
MSCOMM_ER_CDTO	1007	Carrier Detect Timeout beim Versuch, ein Zeichen zu senden (siehe Eigenschaft CDTimeout).
MSCOMM_ER_RXOVER	1008	Empfangspuffer-Überlauf , d. h. der Puffer ist voll.
MSCOMM_ER_RXPARITY	1009	Hardware hat einen Paritätsfehler festgestellt.
MSCOMM_ER_TXFULL	1010	Sendepuffer-Überlauf, d. h. der Puffer ist voll.

Fehlercode-Konstanten

PDQComm for Windows (ZOSCHKE DATA Bestell-Nr. 892)

Noch leistungsfähigere Möglichkeiten zur seriellen Kommunikation mit Visual Basic.

PDQComm for Windows von **Crescent** ist eine Toolbox für die serielle Kommunikation mit Microsoft Visual Basic for Windows. Als Kernstück enthält diese Toolbox mit **PDQComm.VBX** eine **erheblich erweiterte Version** des in der Visual Basic 2.0/3.0 Professional Edition enthaltenen Kommunikations-Controls **MSComm.VBX**, das ebenfalls von Crescent entwickelt wurde.

Zusätzlicher BASIC-Code ist mit PDQComm so gut wie überflüssig. Zum Beispiel erledigen Sie Dateitransfer mit einer einzigen Programmzeile (*Comm1.Upload = dateiname*). PDQComm for Windows bietet gegenüber MSComm mächtige Erweiterungen. Hier die wichtigsten:

- **Voll kompatibel zu MSComm.VBX**
- **Keine Baudraten-Begrenzung**
- **Vielfältige Terminal-Emulationen**
- **Alle gängigen Datei-Übertragungsprotokolle**
- **Modem-Unterstützung durch ModemWare**
- **Control-C-Quelltexte erhältlich**
- **Ausführliches deutsches Handbuch**

Crescent Division of Progress Corp. - bisher Crescent Software, Inc.

Die wichtigsten ModemWare-Befehle

PlaceCall - Wählt die angegebene Telefonnummer und stellt die Verbindung mit dem Modem am anderen Ende her.

WaitForCall - Wartet auf einen Anruf, antwortet und verbindet mit dem Remote-Modem.

LoadPortSettings - Lädt Port-Einstellungen aus einer anwendungsspezifischen INI-Datei und initialisiert den Port. Bei nicht vorhandenen Einstellungen erscheint ein Konfigurations-Dialog.

LoadModemSettings - Lädt Modem-Einstellungen aus einer anwendungsspezifischen INI-Datei und initialisiert das Modem. Bei nicht vorhandenen Einstellungen erscheint ein Dialog zur Auswahl eines Modems anhand der ModemWare-Datenbank mit mehr als 440 Modems.

WaitFor$ - Wartet eine wählbare Anzahl Sekunden auf den Empfang eines ebenfalls angegebenen Zeichens oder Strings.

WaitForA$ - Dito., jedoch kann eine ganze Reihe von Strings als Stringfeld angegeben werden.

SendModemCmd - Sendet einen Modembefehl und liefert den Ergebniscode.

Hangup - Trennt die Modem-Verbindung.

ScriptRecord - Zeichnet eine Skript-Datei auf.

ScriptPlay - Spielt eine Skript-Datei ab.

:Label - Sprungmarke (z. B. :SpringHierher); Verwendung wie in einer Batch-Datei.

PDQComm: Noch leistungsfähiger dank ModemWare

Mit MSComm und PDQComm hat Crescent Software, das kann man wohl ohne Übertreibung sagen, die Möglichkeiten zur seriellen Kommunikation mit Visual Basic revolutioniert. Bei der seriellen Kommunikation zur DFÜ kommt Modems eine besondere Bedeutung zu. Seit der Version 2.10 verfügt **PDQComm** daher über eine neue Komponente namens **ModemWare**. Die zwei wichtigsten Elemente von Modemware sind die **eingebaute Skriptsprache** sowie die Datenbank mit Initialisierungsdaten von bereits mehr als 440 Modems:

Die ModemWare-Skriptsprache

Die wichtigsten ModemWare-Befehle

Die Modem-Datenbank

Bei der **Skriptsprache** in ModemWare handelt es sich um Befehle, um die Kommunikation per Modem zu vereinfachen und von Schreiben typenspezifischen Programmcodes zu entlasten. Unter anderem gibt es Kommandos zur Modem-Initialisierung, zum Anrufen, zum Warten auf Anrufe und zum Trennen der Verbindung. Ein Befehl zum Warten auf den Empfang bestimmter Zeichenketten der Gegenseite ist vorhanden wie die Möglichkeit, Label zu definieren, die sich ähnlich wie in Batchdateien anspringen lassen. Ohne daß ein Anwender ein Terminalfenster zu sehen bekommt, könnte sich ein Programm mit PDQComm und einem ModemWare-Skript in eine Mailbox einwählen, um nach dem Einloggen mit Namen und Paßwort EMail, eine Datei oder Aktienkurse herunterzuladen und sich anschließend wieder auszuklinken.

Dank *Aufzeichnungsmöglichkeit* muß man ModemWare-Skripts nicht unbedingt selbst schreiben: Durch Abspielen aufgezeichneter Skripts, die sich natürlich auch editieren lassen, können einmal manuell durchgeführte Schritte dann automatisch ablaufen.

Die Skriptsprache wurde übrigens in Visual Basic geschrieben, damit sie sich bei Bedarf leicht modifizieren und ergänzen läßt. Ähnlich wie VBA im deutschen Excel 5.0 ließe sich etwa eine Version mit deutschen Befehlen erstellen.

Die **Modem-Datenbank** in ModemWare trägt der Tatsache Rechnung, daß sich die am Markt befindliche Modem-Vielfalt nicht nur im Funktionsumfang unterscheidet, sondern auch hinsichtlich ihrer Initialisierungs-Strings und weiterer Modem-interner Befehle. Da hier kaum ein DFÜ-Programmierer den Überblick haben kann, wurden für PDQComm die Daten von mehr als 440 Modems zusammenzutragen und in einer vom Anwender erweiterbaren Datenbankdatei gespeichert (ein Hilfsprogramm hierzu ist im Quelltext enthalten).

Die gespeicherten Daten enthalten Fabrikat und Typenbezeichnung, höchste unterstützte Baudrate, Initialisierungs-Strings sowie die Befehlscodes für Attention, Hangup, Busy, Dial und Reset. Diese Informationen werden von allen ModemWare-Befehlen, die mit diesen Angaben arbeiten, verwendet.

Baudraten: Mit PDQComm nach oben offen

Die maximale Baudrate wird bei PDQComm nur durch die Fähigkeiten des Windows-Treibers COMM.DRV begrenzt; die Grenzen stecken also Windows und dessen Treiber - nicht das Control.

Vermeidung von EOF-Problemen

Bei PDQComm gibt's keine Empfangs-Überlauffehler nach Empfang eines EOF-Zeichens (ASCII 26).

Kompatibilität ist Trumpf

Das **PDQComm**-Control wickelt sämtliche serielle Kommunikation über das **Windows API** ab und macht sich die Windows-Kommunikationsfunktionen zunutze. Daher arbeitet PDQComm mit sämtlicher serieller Port-Hardware zusammen, die zum Windows-Strandard kompatibel ist. Um die Ports für Windows-Anwendungen zugänglich zu machen, liefern die meisten Anbieter erweiterter serieller Schnittstellenkarten zu ihrer Hardware eigene Windows-Treiber. Sofern Windows diese Hardware erkennt, ist auch PDQComm in der Lage, darauf zuzugreifen.

Darüberhinaus unterstützt PDQComm for Windows jede Baudrate, die von der Hardware und vom Treiber zugelassen wird.

Terminal-Emulationen in PDQComm

- PDQComm for Windows bietet **eingebaute Terminal-Emulationen**: Wird die Eigenschaft "Emulation" auf einen anderen Wert als 0 eingestellt, wird aus dem Control ein Terminal-Fenster, das wahlweise die Emulationen **TTY, ANSI, VT-100** und **VT-52** unterstützt. Diese zeichnen sich durch folgende Merkmale aus:
- **Einstellbarer Scrollback-Puffer** - Dieser Pufferspeicher erlaubt das Betrachten der zuletzt empfangenen Daten (bis zu 64K). Text kann per Maus markiert und aus dem Terminal-Screen herauskopiert werden werden; umgekehrt kann Text aus der Zwischenablage eingefügt und zum COM-Port geschickt werden.
- **Display Capture** - Einfangen der angezeigten Daten und Abspeichern in einer Capture-Datei (wahlweise mit oder ohne Filterung der Daten zur Anzeigesteuerung).
- **Freie Schriftwahl** - Alle VB-Fonteigenschaften wie FontName, FontSize, FontBold, FontItalic und FontUnderline lassen sich einstellen.
- **Farbfilter** für Graustufen-Darstellung (z. B. LCD) und Monochrom-Bildschirme.
- **Automatische Verarbeitung von Tastendrücken und empfangenen Zeichen**, sodaß das Terminal-Programm keinerlei zusätzlichen Programmcode erfordert!
- **Schnelle Bildschirmaufbereitung**, da PDQComm ohne Subclassing eines anderen Controls auskommt.

Datei-Übertragungsprotokolle in PDQComm

PDQComm for Windows bietet zahlreiche Datei-Übertragungsprotokolle, und zwar **XModem-ChkSum, XModem-CRC, XModem-1K, YModem-Batch, YModem-G, ZModem, Compuserve B+** und **Kermit**. Näheres siehe Datei-Protokolle.

Die Protokolle zeichnen sich durch folgende Merkmale aus:

- Einstellung komplett über Eigenschaften - Kein Programmcode erforderlich!
- Eingebaute Dialogbox zur Statusanzeige der Übertragung.
- Unterstützung von sowohl Einzeldatei- als auch Batch-Übertragung.
- Die Übertragung geschieht im Hintergrund, ohne im vom Windows vorgegebenen Rahmen irgendwelche laufenden Programme zu beeinträchtigen.
- Nach Initialisierung der Übertragung wird die Kontrolle unmittelbar an das aufrufende Programm zurückgegeben, sodaß dieses sofort wieder andere Aufgaben erledigen kann. Die Übertragung selbst muß also nicht abgewartet werden.
- Das OnComm-Ereignis erlaubt die Beobachtung der empfangenen Daten; die eingebaute Dateitransfer-Dialogbox kann automatisch den Fortschritt der Übertragung anzeigen.
- Der gesamte Übertragungs-Code wurde in C geschrieben (Quelltext ist verfügbar).

Warum auf jeden Fall PDQComm for Windows einsetzen?

- Für die serielle Kommunikation mit Visual Basic gibt es verschiedene Alternativen. Es gibt jedoch einleuchtende Gründe, sich für **PDQComm for Windows** zu entscheiden:

- PDQComm for Windows ist ein Superset, also eine Übermenge des MSComm-Controls.

- Crescent Software hat **MSComm** für **Microsoft** entwickelt, und PDQComm for Windows ist sozusagen **MSComm's großer Bruder**.

- Sie bauen also gegebenenfalls auf Ihre bereits gesammelten Erfahrungen auf. Für MSComm geschriebener Quelltext läuft auch mit PDQComm for Windows.

- Die mit PDQComm gelieferten Protokolle für binäre Dateiübertragung wurden in C geschrieben, also nicht einfach als Basic-Code implementiert.

- Genau wie QuickPak for Windows wird auch PDQComm for Windows konsequent weiterentwickelt. PDQComm-Erfinder Dave Cleary gehört inzwischen fest zum Crescent-Entwicklungsteam.

- Crescent Software widmet sich dem Thema Kommunikation sehr eingehend und hat in diesem Bereich auch für die Zukunft noch so einiges vor.

- Beispielsweise schuf Crescent Software für die PDQComm die Scriptsprache ModemWare; für die hierzu gehörige Modem-Datenbank wurden Initialisierungs- und weitere relevante Daten von nahezu 450 Modems gesammelt.

- Ein OLE Custom Control (OCX) für 16/32 Bit ist bei Crescent bereits in der Entwicklung; hierfür wird zu gegebenem Zeitpunkt Upgrade-Möglichkeit angeboten. PDQComm ist also zukunftskompatibel.

- Bei Bedarf steht sogar eine PDQComm-Bibliothek für DOS-Anwendungen zur Verfügung, die sich mit QuickBASIC, BASIC PDS und VB/DOS einsetzen läßt.

PDQComm for Windows: Lieferumfang

Programmdiskette
Diskette mit PDQComm-Control und diversen Beispielen, einschließlich eines kompletten komfortablen Terminal-Programms im Quelltext sowie umfassender Windows-Online-Hilfe zum PDQComm-Control und zu ModemWare (einschließlich separater Skriptsprachen-Hilfe für eigene Anwendungen).

Die C-Quelltexte des Controls können von registrierten Anwendern bei Bedarf kostenlos angefordert werden.

Ausführliches deutsches Handbuch
Zu PDQComm for Windows gehört ein ausführliches deutsches Handbuch. Es liefert Ihnen einführende Informationen zur seriellen Kommunikation und erklärt Modems, serielle Kabelverbindungen, Port-Parameter und UARTs. Alle wichtigen AT-Modembefehle werden im Detail erläutert; für alle Terminal-Emulationen zeigen Tabellen die jeweils unterstützten Codes. Hinzu kommen wertvolle Tips und Tricks zur seriellen Datenübertragung, Modem-Steuerung usw.

Bei ZOSCHKE DATA registrierte Benutzer von PDQComm 2.x mit englischem Handbuch können das deutsche Handbuch nachträglich erwerben (Seriennummer und Kaufnachweis erforderlich).

Lizenz
Zusammen mit seinen ausführbaren Anwendungen darf der Lizenznehmer das PDQComm-Control ohne Zahlung von Lizenzgebühren verbreiten. Dies gilt auch für die Skriptsprachen-Hilfedatei, für die ZOSCHKE DATA registrierten Anwendern auf Wunsch auch eine deutsche Fassung zur Verfügung stellt.

Technischer Support
Die ZOSCHKE DATA GmbH leistet ihren registrierten Anwendern für *PDQComm for Windows* und alle weiteren angebotenen Produkte deutschen Update-Service und kostenlosen technischen Support. (Für den technischen Support zu *MSCOMM.VBX* ist die Firma *Microsoft* zuständig).

PDQComm for Windows eignet sich gleichermaßen für *Microsoft Visual Basic 2.0* und *3.0* in der Standard- und in der Professional-Ausführung.

Weitere PDQComm-Versionen

Weitere PDQComm-Versionen

- Für **DOS-BASIC** (QB, PDS und VB/DOS) ist eine DOS-Version von PDQComm verfügbar (ebenfalls mit deutschem Handbuch; für Benutzer der Windows-Version steht die DOS-Version als vergünstigte Zusatzlizenz zur Verfügung.
- Eine **OCX-Version** für Entwicklungssysteme, die solche OLE-Controls unterstützen, ist für das erste Quartal '95 angekündigt.

PDQComm 2.x for Windows: Pressestimmen

"Zusammenfassend läßt sich feststellen, daß PDQComm 2.0 für jeden, der sich mit dem Dateitransfer über die serielle Schnittstelle beschäftigt, ein Muß darstellt."
Computer Persönlich 5/94

"Das umfangreiche Beispielprogramm realisiert ein Telekommunikationsprogramm, das sich hinter kommerziellen Anwendungen keinesfalls verstecken muß. ... Das Handbuch ist ansprechend und klar gegliedert. Es liefert zusätzlich eine verständliche Einführung in die technischen Hintergründe der Datenkommunikation."
DOS Extra 1/94, Alles rund um BASIC

"Mit PDQComm for Windows hat Crescent Software ein Kommunikations-Tool geschaffen, das Visual Basic auf den Leib geschneidert ist und an Komfort und Umfang seinesgleichen sucht. Die Art und Weise, wie in PDQCOMM.VBX selbst komplexe Abläufe wie Terminalemulationen als Eigenschaften realisiert wurden, macht dieses Custom Control allen uns bekannten DLL-Lösungen überlegen."
Basic Professionell 3/94

Schon über Version 1.x urteilte Computer Persönlich:
"Alle Library-Funktionen [in PDQComm for Windows] sind nicht nur gut dokumentiert, sondern auch leicht handhabbar."
Computer Persönlich 8/92

Unter dem Namen **Basic Professionell** gibt der **Steingräber Fachverlag** eine deutsche Fachzeitschrift heraus, die sich ausschließlich Visual Basic und verwandten Basic-Dialekten widmet. Zu jeder Ausgabe erscheint eine separate Heftdiskette mit allen Listings und weiteren interessanten Dateien; in unabhängigen Abständen erscheint außerdem eine Basic-CD-ROM. Desweiteren werden vom Herausgeber Basic-Entwickler-Konferenzen organisiert.

Ein kostenloses Probeheft kann beim Steingräber Fachverlag unter der CompuServe-Nummer **100111,3245** angefordert werden, indem Sie sich auf diesen Artikel (VBCOM.HLP) beziehen.