EVALUATING RS-232 COMMUNICATION PACKAGES

Frank da Cruz, Christine Gianone

Columbia University Center for Computing Activities
New York, N.Y. 10027

December 1987 [1]

In most places where there are computers, nobody questions the need for data communication. The question is more likely to be how best to do it. In theory networks can do the job, typically PC LANs, perhaps interconnected by gateways to mainframe networks. But this arrangement presumes that equipment was purchased according to some consistent plan, with networking in mind. In practice, most organizations have a patchwork history of computer acquisition, resulting in a hodgepodge of PCs, minis, and mainframes for which compatible networking options are not available or affordable. Often, the only device these systems share in common is the humble RS-232 asynchronous communication port. Even when comprehensive networking solutions exist, the cost of attaching hundreds or thousands of PCs, at $500-$1500 per PC, to a Token Ring or Ethernet can be daunting. And those who want to dial in from home, or dial out to external services, must still be accommodated.

For these reasons, many organizations choose to connect PCs to their networks through use of RS-232 communication packages like Crosstalk, Blast, Relay Gold, Smartcomm, VTERM, Kermit, PC-Talk, ProComm, Red Ryder, HyperACCESS, or ASCII Pro. Hundreds of these products permeate the marketplace, and, if selected intelligently, they can fill the gaps in an organization's network effectively and inexpensively. The cost for PC-resident data communication programs typically ranges from $20 to $500 per PC, with some exceptions (some are free, others cost more). And that makes these programs more affordable than most PC network connections.

Evaluating RS-232 communication packages can be a complicated process. The Cost must be weighed carefully against the needs of your organization. Reviews and surveys of communication packages appear frequently in the popular computer publications, and they can help. But surveys consisting mostly of charts in which, say, 100 popular software packages are compared on the basis of, say, 20 arbitrarily chosen features, are necessarily superficial. And many articles concentrate on frills and conveniences while giving short shrift to central issues like connection establishment and maintenance, terminal emulation, and file transfer. In this article we attempt to present the features of RS-232 communication packages in a way that will help you to assess their relative importance for yourself or your organization.

---

[1]This is the original manuscript of the article that was published in Data Communications Magazine as "Shopping for Software That Lets PCs Chat With Mainframes", December 1987, pp.155-170.

WHAT IS AN RS-232 COMMUNICATION PACKAGE?

RS-232 communication packages are software programs that run on personal computers like the IBM PC, IBM clone, Apple II, or Macintosh, and communicate asynchronously with remote computers through RS-232-C serial ports, either directly or through modems. This distinguishes them from specialized products that emulate synchronous terminals in IBM mainframe, Wang, or similar environments, for which special adapters (like an Irma board) and connections (like coax) are required, and which won't be discussed here.

There are two fundamental aspects to any PC communication package. One is terminal emulation, which allows you to connect your PC to a mini or mainframe as a terminal in order to conduct a timesharing session or to access an application on a central, shared system. The other is data transfer, which lets you exchange information between your PC and a mini or mainframe (or another PC). For terminal emulation, PC-resident software is the only requirement. For error-free data transfer, however, a companion program is required on the other computer. The cost for commercial mini- or mainframe-based communication programs runs much higher than PC versions, typically ranging from $1000 to $100,000 (again, with exceptions).

USER INTERFACE

The most immediately striking aspect of any program is the "user interface": the prompts, commands, menus, function keys, etc, with which you make your wishes known to the program, and the displays with which it makes the results known to you.

There are many styles of user interface: command line, interactive prompt and command, menus and arrow keys, mice and windows. The fundamental tradeoff is ease of learning versus ease of use. Ease of learning is important if many people will be using the package infrequently or if there is rapid personnel turnover, so that relatively little time need be "wasted" in learning and training. A user interface designed for ease of learning presents you with all the choices in menus. The penalty is that you are always presented with menus for everything, which slows you down needlessly once you have become expert. At the other extreme are programs that favor the expert, providing only terse and cryptic commands, sometimes with no way for a novice to get help at all, short of reading the manual. A compromise, "menu on demand", lets the expert issue rapid, terse commands, while still allowing the novice to see a menu at any point by entering a special help key.

There is an oft-neglected aspect of the user interface that falls into the "ease of use" category: can the package be used by people with disabilities like motor impairment, blindess, or deafness? If you can depress only a single key at a time, how can you enter complicated Ctrl-Alt-Shift key combinations? If your PC is connected to an ASCII-oriented speaking device or a Braille terminal, how can you decipher multicolor animated graphics screens? If you can't hear, how do you know when the package is beeping or whistling at you to signal some important event? Often, the fancier the user interface, the less it lends itself to use by the disabled.

Another item of minor importance, but whose absence can be a nuisance, is the ability to access system functions without actually leaving the program. If you want to change directories, list files, display a file, or delete a file, you should not have to exit the communication program, and then restart it afterwards. This can be time consuming, especially on floppy-disk-based systems, and even more so when settings -- or the connection itself -- must be reestablished.

Commercial communication packages tend to place great emphasis on the appearance and style of the user interface, primarily for marketing reasons. But for most people, the user interface should not be a key factor in evaluating a product. It only lets you specify the real work to be done, and it should take up a relatively small proportion of the total time you spend with the package. Ultimately, it is much more important to know whether the product can perform the required tasks, and how well.

CONNECTION ESTABLISHMENT AND MAINTENANCE

Perhaps the most important aspect of any communication package its set of mechanisms for establishing a connection, matching communication parameters to the communication medium and the system on the other end, monitoring the connection once established, and breaking the connection.

COMMUNICATION PARAMETER SETTINGS

Any communication program should allow you control over communication parameters like bits per second, parity, duplex, flow control, and the number of data, start and stop bits per character. Each of these parameters is important, and the lack of any particular option may prevent you from communicating satisfactorily with another computer. Before you select a communication package, you should be sure it supports all the communication parameters and settings required by all the computers you need to communicate with.

For example, most DEC minis and mainframes employ XON/XOFF full duplex flow control to prevent data overruns; if your PC communication package does not support XON/XOFF, then data transferred between it and the DEC system could be lost. IBM mainframe ASCII TTY linemode connections, on the other hand, are half duplex and exercise a line turnaround handshake discipline: if you transmit to the IBM mainframe before it has given you permission (by sending a special "handshake" character, such as Control-Q), it will not accept your data. Certain popular mainframes and minis, as well as public data networks like Telenet and Tymnet, use even, odd, or mark parity, and will not recognize your characters unless the right parity is applied. And if your package cannot distinguish parity bits from data bits, the wrong characters will be displayed on your screen. Table 1 shows typical RS-232 communication parameters for various systems.

Computer	Front End	Duplex	Flow Control	Parity	Terminal
Data General MV	None	Full	XON/XOFF	None	Dasher
DEC PDP-11	None	Full	XON/XOFF	None	VT52,VT100
DEC VAX	None	Full	XON/XOFF	None	VT52,VT100
DECSYSTEM-20	PDP-11	Full	XON/XOFF	Even	VT52,VT100
Honeywell DPS8	DN335	Half	XON Handshake	None	VIP7300,7800
HP-1000	None	Full	ENQ/ACK	None	HP262x
HP-3000	None	Half	XON Handshake	None	HP262x
IBM 370 Series	3705 TTY	Half	XON Handshake	Mark	TTY
IBM 370 Series	7171 P.E.	Full	XON/XOFF	Even	Various*
Prime minis	None	Full	XON/XOFF	Mark	TTY, PS300

P.E. = 3270 Protocol Emulator, TTY = ASCII Linemode Connection.
*Delivered with support for 13 popular terminals, configurable for more.

Table 1: Typical Communication Parameters

Some communication packages support only a limited range of transmission speeds. For instance, they may be designed to work only for dialup connections at speeds up to 1200 or 2400 bits per second (bps). If you should ever need to connect two computers directly, you should not be artificially limited to such low speeds. Even if dialups are the only connections you will ever make, you should be aware that new modems are appearing on the market that operate at speeds in excess of 9600 bps on ordinary voice-grade phone connections. For these reasons, any communication package should be able to operate at 9600 bps or higher. 9600 bps is the highest speed supported by most minis, mainframes, and front ends today (some support 19,200 bps). Micros like the IBM PC/AT and the Macintosh, however, can drive their RS-232 ports to speeds of 38,400 bps or higher, and two such PCs connected back to back can actually transfer data at these speeds. But the higher the speed, the more important it is to have an effective flow control mechanism supported by the systems on each end of the connection.

MODEM CONTROL

Unless your computers are hardwired together with dedicated lines, you probably use asynchronous dialup modems to establish connections with other systems. Modems communicate special control information with the PC via RS-232 modem signals like DTR, DSR, and CD (see Table 2). Most communication packages can control and monitor these signals to detect when the connection is broken, or to break the connection and hang up the phone. If you use your package interactively, modem control is largely superfluous because you will notice when the connection is broken, or you will hang up the phone yourself when you are done with it. For unattended operation, on the other hand, modem control is important in avoiding the excessive phone bills that could result when the communication package does not notice that connections break and leaves the phone off hook all night.

FG	Frame Ground
TD	Transmitted Data, from PC to modem
RD	Received Data, from modem to PC
RTS	Request To Send, from PC to modem, used with half duplex modems
CTS	Clear To Send, from modem to PC, ditto
SG	Signal Ground
DSR	Data Set Ready, indicates modem is in data transmission mode
CD	Carrier Detect, indicates that modem is connected to other modem
DTR	Data Terminal Ready, tells modem that PC is ready to communicate
RI	Ring Indicator, tells PC that the phone is ringing

Table 2: RS-232-C Asynchronous Modem Signals

Modems may be either external or internal. External modems are controlled in a consistent way according to RS-232-C, and rarely pose a problem to communications software. Although generally more expensive, external modems are interchangeable between different computers. Internal modems, on the other hand, are built specifically for certain computers, and sometimes require special handling by the software. A particular software package will not necessarily operate correctly with a specific internal modem (check with the software vendor!). And, of course, two modems that are to communicate must support the same modulation techniques and speeds.

Some packages are designed to be used only with modems, and don’t work properly when two computers are connected directly by a cable, unless certain modem signals are "faked" by cross-connections or jumper wires within the cable connectors. Such a fakeout cable is called a "null modem" or "modem eliminator" (Figure 1), readily available from any computer supply house, and also available as an adapter that you can connect to your modem cable. But different systems may require different signals connected in different ways, so be prepared for some experimentation (a breakout box will help). Your communication software package can relieve you of the tinkering if it can be configured to ignore modem signals altogether when you connect two computers directly.

```
              True Null Modem                    Minimal Null Modem
                 (8 wires)                           (4 wires)

             FG ----------- FG                  FG ----------- FG

             TD ----\ /---- TD                  TD ----\ /---- TD
                     X                                  X
             RD ----/ \---- RD                  RD ----/ \---- RD

            RTS ----\ /---- RTS                RTS -+       +- RTS
                     X                              |       |
            CTS ----/ \---- CTS                CTS -+       +- CTS

            DSR -+       +- DSR                DSR -+       +- DSR
                 |       |                          |       |
             CD -+--\ /--+- CD                 CD  -+       +- CD
                     X                              |       |
            DTR -+--/ \--+- DTR                DTR -+       +- DTR
                 |       |                          |       |
             RI -+       +- RI                  RI -+       +- RI

             SG ----------- SG                  SG ----------- SG
```

Figure 1: Null Modems

DIALER CONTROL

Many PCs are connected to the outside world only by telephone. "Smart modems", like those manufactured by Hayes, are able to dial the phone for you if they receive commands in the right format from the PC. This means that your communication package must understand the dialing language of the modem. Although the Hayes "AT" language has become a de-facto standard, not all autodial modems conform to it (prominent counterexamples include DEC DF-series modems, and selected models from Racal-Vadic, US Robotics, and Ventel). Be sure your modem's dialing language is supported by your communication package.

Dialing software simplifies connection establishment, hiding the details of the dialing language from you. You tell the program what number to call, and let the software handle the details. Some packages go a step further and include a phone directory so that you don't even have to remember phone numbers, just the names of the places you want to call.

Dialer control and phone directories fall into the frill category for most users. It's not much more trouble to type "ATDT7654321" than it is to type "dial fred". For unattended operation, however, automatic dialing is an essential feature. If a dial command is lacking, then the same function can be performed by a script, described below.

DEBUGGING COMMUNICATION PARAMETERS

Often, we can only guess what the right combination of speed, stop bits, parity bits, data bits, duplex, and flow control might be for a particular connection. What if we guess wrong? What tools does the communication package give us to pin down the offending parameters?

Obviously, if the package lets us set these parameters independently, we can vary them until the connection works, but the combinations could be endless. Your communication package should include debugging tools to reduce the guesswork, like special display or logging of received characters, preferably including their 8-bit numeric values. If examination of the log reveals a byte with a numeric value of 193 (= 11000001 binary) where you would expect an ASCII "A" (= 01000001 binary), then you probably should be looking for 7 data bits with odd or mark parity rather than 8 data bits with no parity.

A good communication package will also include a troubleshooting guide, like the one in Table 3, in which you can look up symptoms and find the corresponding diagnoses and prescriptions.

SYMPTOM	POSSIBLE CAUSE	CURE
Blank, dark screen.	PC turned off.	Turn on PC.
Total garbage on screen.	Wrong speed.	Try another speed.
Spurts of garbage on screen.	Noise.	Hang up and redial.
Uniform mixture of good and bad characters on screen.	Parity.	Select a different parity.
Typed characters appear twice.	Duplex.	Select full duplex.
Typed characters don’t appear.	Duplex.	Select half duplex.
Random gaps in screen text.	No flow control.	Use flow control, or a slower speed.

Table 3: Sample Entries from a Troubleshooting Guide

SAVING COMMUNICATION PARAMETERS

Once you have discovered the proper settings for communicating with a particular machine, you will want to have some way of saving them. To alleviate the tedium of setting five or ten communication parameters each time you connect to a given system, the package should allow settings to be collected together into "configurations" which may be saved under mnemonic names.

Some packages are delivered with a set of configurations for popular dialup services like Dow-Jones, Compuserve, MCI Mail, The Source, etc. These built-in configurations shield you from having to know anything about data communication parameters. But when you must establish a connection to a system the package doesn’t know about -- like from your PC at home to the mainframe at work -- you should be able to manipulate the communication settings yourself and save them for future use. Once you’ve determined the appropriate settings for, say, your company’s DEC VAX, IBM 3090, Harris 800, plus your local Telenet PAD, you only need mention the associated configuration name to set all the corresponding parameters at once.

SCRIPT LANGUAGE, UNATTENDED OPERATION

Just as a communication package may remember your communication settings or phone numbers for you, it can also allow repetitive interactive tasks, like login sequences, to be automated by means of "scripts". Scripts are little "programs" that look for specific outputs from the remote computer and provide appropriate responses. When the package, or the underlying system, allows a script to be executed at a predesignated time, then it is possible to carry on a canned dialog with no human operator present. For instance, you might program your PC to "wake up" at midnight, set the proper communication parameters and dial up your office mini, log in, deposit the day's transactions, fetch and print the day's mail, log out, and hang up. Script languages vary from the primitive and cryptic to full-blown programming languages complete with variables and conditional branching. Figure 2 shows a simple script for dialing a Hayes modem to establish a connection to a Unix system and then logging in. It illustrates how a script can be used in place of built-in dialer control.

```
set speed 1200                    Bits per second
set parity none                   Parity
output AT\13                      Wake up the modem
input OK                          Look for its "OK"
output ATD7654321\13              Give modem's dialing command
input CONNECT                     Look for desired response
pause 1                           Wait a second
output \13                        Send a carriage return
input login:                      Look for login prompt
output chris\13                   Send user ID, followed by carriage return
input Password:                   Look for password prompt
echo Connecting to Unix System...
echo Please type your password:
connect                           Let the user take over
```

Figure 2: Script Language Example

The "output" commands send the indicated text strings to the Unix system ("\13" is a code for carriage return), and the "input" commands search the incoming data for the indicated strings. If any of the input commands fail, the script is automatically terminated. This is an important feature of a script language. Suppose, for instance, you have a nightly script which sends your day's work to a mainframe and then deletes it from the PC's hard disk. If the data could not be successfully transmitted, then you certainly don't want the script to forge ahead stubbornly and destroy all your work.

Scripts are essential for unattended operation, and they are also useful in setting up procedures for relatively unskilled operators such as data entry clerks. For the typical interactive user, however, scripts are a minor convenience rather than a necessity.

TERMINAL EMULATION

The PC has increasingly replaced the terminal in many organizations. In addition to its other capabilities, a properly programmed PC can also act like ("emulate") a terminal, so that you can use it to conduct a dialog with a remote computer. Your keystrokes are sent out the communication port, and characters that arrive at the port are displayed on the screen. On half duplex, local echo connections, your keystrokes are also displayed on the screen. On full duplex connections, terminal emulation can be a tricky business because characters may arrive at the port at the same time that you are typing; communication programs vary in their ability to handle both events at once, especially at higher speeds.

It should be stressed that terminal emulation does NOT provide any sort of automatic error control, any more than a real terminal would. Bare characters are sent back and forth with absolutely no error recovery mechanism. If a package claims to supply error-checking data transfer, you should understand that this claim applies to its file transfer functions and not to its terminal emulator. A noisy telephone line would probably leave garbage on your screen during terminal emulation even though files could be transferred successfully.

In addition to sending and displaying characters, a terminal emulator also attempts to imitate the repertoire of special effects of some particular real ASCII video display terminal, such as the DEC VT100 series, the IBM 3101, the Televideo 920, the ADM3A, etc. This means that the program responds to screen control sequences sent by the host just as the real terminal would. For example, the ASCII sequence "ESC [5 ; 7 H" sent to a DEC VT100 positions the cursor at row 5, column 7; "ESC [0 J" clears the screen, and so a PC programmed to emulate a VT100 would understand the same sequences and perform the same actions. When emulating a terminal, the package should also provide some mapping between the terminal's function keys and the PC's, so that they transmit the same sequences. If the VT100 PF1 key sends "ESC O P", then the IBM PC's F1 key might be programmed to send the same sequence.

Today's video display terminals possess a formidable array of features for tabbing, highlighting, partitioning the screen, erasing and inserting text, positioning the cursor, drawing figures, changing colors, switching character sets, activating printers, etc, all controlled by host-transmitted escape sequences. A package may emulate such a terminal completely, or it may emulate a "subset" of its functions. Some terminals have features that cannot be emulated by certain PCs. For example, the DEC VT100 allows switching between 80- and 132-column modes, but an IBM PC can only display 80 columns. To get 132 columns on a PC, a special board may be needed. Another example is the VT100's "smooth scrolling" feature, which allows a file to glide slowly along the screen -- the DEC Rainbow can do this, while the IBM PC cannot.

Emulation should be complete enough to allow you to access any desired software on the host which expects to control the appearance of the terminal's screen; full-screen text editors like EDT on VAX/VMS or GNU EMACS on a UNIX system are good tests. Another is IBM 3270 protocol emulation as performed by the IBM 7171 or other protocol converter. If emulation is not complete, you will see fragmented and jumbled screens, characters or lines overwriting each other, mysterious gaps and transpositions.

Terminal emulation is a very important function for people who engage in a lot of interactive dialog with a remote system, especially when screen control is involved. In this case, it is essential that the communication package be capable of emulating a terminal that the remote system supports, such as a DEC VT100 or -200 series with a DEC VAX/VMS system, a Data General Dasher with DG minis, etc. (see Table 1). Terminal emulation is less important for brief or non-interactive encounters, such as occasional sessions primarily for file transfer.

KEY REDEFINITION AND CHARACTER TRANSLATION

Since your PC keyboard may have a different layout than the emulated terminal, you may want to "move" the misplaced keys to their familiar locations (no, you can't use pliers for this). For instance, the Escape (ESC) key (important to much host-resident software) is notoriously mobile, appearing in many different locations even on PCs from the same maker (IBM and DEC spring to mind). If you're used to finding ESC immediately left of the "1", but your PC has "‘" (accent grave) in that position, you could redefine "‘" to transmit ESC (and vice versa). Similarly for function keys: the VT100 PF keys are on the right, whereas the IBM PC's F keys are on the left; some VT100 users may find it more convenient to assign the PF keys to the PC's numeric pad.

A package might also allow you to assign any arbitrary character string to a key, so that you could transmit commonly typed items like your name or login sequence with a single keystroke. Such many-to-one assignments are called "keyboard macros", and there are limits to the number of characters which may be represented by a single key.

Key redefinition is important if you switch frequently among terminals and PCs with different keyboard layouts; it means you don't have to retrain your fingers each time -- a blessing for touch typists. It is also helpful when switching the same PC between different hosts. If you are used to typing the Backspace key to erase a character, but one host uses ASCII Rubout for this function while another uses Control-H, you can assign the suitable character to the Backspace key.

Like communication settings, key definitions might take you some time and experimentation to perfect. Once you have configured your keyboard satisfactorily, you should be able to save your definitions for future use.

CHARACTER SETS

The ability to handle European and non-Roman character sets (keyboard input as well as screen output) is important for those who deal in languages other than English. It is common practice in Germany and Scandanavia, for instance, to assign umlaut, slashed, or circled vowels to the ASCII bracket positions. PCs and host computers must agree upon these conventions in order for characters to be displayed as intended, rather than in Anglo-American ASCII. Translation of outbound and arriving characters is therefore an important function of the communication package. To be totally general, the package should not be restricted to 7-bit ASCII, but should allow for 8-bit international character sets, in line with ISO Recommendations 2022, 6937, et al.

TEXT SCREEN MEMORY

A special advantage of emulating a terminal on a PC is that the PC may surpass the capabilities of the terminal. The PC's memory may be used to hold hundreds of lines that have scrolled off the top for later recall. Current or previous screens may be dumped to a disk file or printer at the touch of a button.

Screen print, dump, and rollback fall into the convenience category, and yet once you've become used to them, you wonder how you ever lived (or at least, worked) without them. How many times has some important message scrolled off your screen before you could read it? How many times have you typed hundreds of lines of text into a computer that crashes before you could save your work?

GRAPHICS

If your PC has a color monitor, your communication program should be able to set the fore- and background text screen colors. A well-chosen color scheme can reduce "operator fatigue" or, conversely, can jolt you awake during the less exciting hours of your day.

In order to access graphics-oriented applications on your mainframe or mini such as SAS Graph, SPSS Graphics, Plot 10, TELL-A-GRAF, or various CAD packages (not to mention certain dialup shopping services), the communication package must emulate a graphics terminal or standard known to the application, such as Tektronix 4010, 4014 or other model, DEC ReGIS, HPGL, GKS, GDDM, NAPLPS, etc. Graphics terminal emulation is only found in a few communication packages, usually as an extra-cost item, and for certain PCs (like IBM) a special monitor and graphics board are also required.

In recent years, graphics tend to be done directly on the PC by such packages as Lotus, Macpaint, etc. It is normally not possible to connect one PC to another in order to access the remote PC's graphics applications, though certain highly specialized packages do allow this. You cannot expect to run Crosstalk from PC A to PC B, and expect Lotus on PC B to put a color pie chart on PC A's screen. More commonly, the graphics package exists on both PCs and their data files are moved from one computer to another using a file transfer protocol built into the communication package.

FILE TRANSFER

"...transfers your data over phone lines at the speed of light!" was a claim that once appeared in an advertisement for a communication package. While it's true that electricity travels through wires at near light speed, it is not (yet) true that one electron is equivalent to one bit of data. In fact, at the most common speed used for dialup data communication, 1200 bits per second, a single bit is pretty big -- about 150 miles long! A character (generally represented in transmission by 10 bits) is 1500 miles long; two characters, like "OK", would span the American continent.

Spurious advertising claims notwithstanding, transmission speed is a technological issue, but data transfer is a software issue: How to make effective use of the transmission medium? How to smooth over the differences between computers?

There are also several specific areas to watch out for. Can binary files be transferred? Can text file formats be converted to useful form between unlike systems? Can a group of files be sent in a single operation? Can filename collisions be avoided? Can a file transfer be cleanly interrupted?

ASCII VS ERROR-CHECKED PROTOCOLS

Communication packages offer two basic types of data transfer: "raw" and error-checked. The most common "raw" method is usually billed as "ASCII protocol". This means that the data is sent as-is, as ASCII characters, from one computer's communication port to the other. The advantage is simplicity. No special software need be resident on the remote computer, beyond its text editor, or a "type" or "copy" command. The disadvantages, however, explain why error-checked protocols have evolved, and are worth noting. The data sent using the ASCII protocol will be corrupted if there is noise on the communication line. Data will be lost if the receiving computer can't keep up with the sender. Binary (non-textual) files generally cannot be transferred this way since many computers will ignore the "parity bit", or act upon control characters rather than accept them as data: Control-C, Control-S, and Control-Z are frequent culprits. And finally, this method works for only one file at a time.

A refinement of ASCII protocol incorporates XON/XOFF or some other flow control method, to reduce the chances of data loss. In this case, both computers must support the same flow control method, but corruption of the data (including the flow control signals themselves) remains a problem, as does file delimitation and the restriction on binary files.

If you want reliable, correct, and complete transmission of files between computers, then you can't trust the job to ASCII or XON/XOFF "protocol". You'll need a communications package that includes a true error-correcting file transfer protocol. Error-checked data transfer requires cooperating programs on each end of the connection to exchange messages, called packets, according to agreed upon formats and rules, similar to how we behave on the telephone: I dial, your phone rings, you pick up and say hello, I identify myself, we take turns talking and if I didn't understand what you said then I ask you to repeat (and vice versa), then we say goodbye, and then we hang up. And (an important point) we conduct the conversation in the same language. A file transfer protocol operates similarly: the two processes "connect" with each other, identify the files that are being transferred, request retransmission of lost or damaged packets, identify the end of the file, and then disengage from each other.

By the way, the fact that many newer modems provide error correction does not eliminate the need for file transfer software. An error-free data stream from modem to modem does not guarantee correct data from computer to computer. Issues of end-to-end flow control and error correction, file delimitation, and format conversion must still be addressed within the computers themselves.

XMODEM AND KERMIT

Two well known error-checking file transfer protocols are Xmodem and Kermit. Many commercial packages include one or both of these protocols (sometimes alongside their own private, proprietary protocols), but there are also hundreds of public domain or freely sharable Kermit or Xmodem programs. In fact, the major advantage of Xmodem and Kermit is that they are ubiquitous. The protocol specifications are open and public, and large bodies of Kermit and Xmodem software are available. The cost to a large organization for these programs is minimal, compared with the per-CPU licensing fees required for commercial packages. Furthermore, chances are greater that a Kermit or Xmodem program will exist for any given computer.

In the case of Kermit programs, source code is included, which encourages their adaptation to a wide range of systems. Non-commercial Kermits can be had for more than 250 different machines and operating systems, ranging in size from the smallest micro to the largest supercomputer, and Kermit is included (at no extra charge) in about 100 different commercial software packages. And, according to recent announcements from Telebit and AST, Kermit protocol is even beginning to find its way into silicon. Xmodem is also available for a wide variety of computers, but it was designed primarily for micro-to-micro links. It is most widely known by its commercial implementations, as a fixture in programs like Crosstalk.

Kermit, Xmodem, and other error-checking protocols are not equivalent. Kermit won't talk to Xmodem, and vice-versa. Each must be evaluated according to several criteria: Is there a version of the protocol available for all the systems that must communicate? Can the protocol accommodate all the communications parameters required for the systems, and for the communication medium? Is the performance acceptable? Is the software affordable?

Xmodem is more properly called the Christensen protocol after its designer, Ward Christensen, who originally intended it only for communication between CP/M micros. Ward put his original 1977 MODEM program into the public domain, and it was modified by others over the years, and some protocol features were added, resulting in protocol variants with names like MODEM2, MODEM7, XMODEM, YMODEM, ZMODEM, etc.

The Kermit file transfer protocol was originally developed in 1981 at the Columbia University Center for Computing Activities for CP/M, MS-DOS, the DECSYSTEM-20, and IBM mainframes with VM/CMS; that is, for use in the micro-to-mainframe environment. It was shared freely with other institutions, with sources and documentation included. Everyone was, and is, permitted and encouraged to copy and share, to make improvements, and to contribute new versions.

Kermit and Xmodem both transfer files between computers in blocks of data, or packets. Both protocols require a program running on each computer to compose, send, read, decipher, and act upon the packets. Each packet is error-checked through the use of calculated checksums, and retransmission is requested when packets have incorrect checksums. Deadlocks are broken by timeouts and retransmission. Missing or duplicate packets are caught using packet sequence numbers. Both protocols are half-duplex stop-and-wait: the next packet is not sent until the current packet is acknowledged. Kermit and Xmodem packets are illustrated in Figure 3.

```
Xmodem:
+-------+-------+-------+-----------------+-------+
| SOH   | BLOCK |-BLOCK | DATA (128 bytes) | CHECK |
+-------+-------+-------+-----------------+-------+

All fields are 8-bit binary:

SOH     is ASCII Control-A (SOH, Start of Header).
BLOCK   is the 8-bit binary "block" (packet) number, 1-127 (recycles).
-BLOCK  is 255 minus the block number (1's complement of block number).
DATA    is exactly 128 bytes of unencoded 8-bit data (a CP/M disk block).
CHECK   is an 8-bit binary checksum.

Kermit:
+-------+-------+-------+-------+---------+-------+
| START | LEN   | SEQ   | TYPE  | DATA.... | CHECK | <cr>
+-------+-------+-------+-------+---------+-------+

Each Kermit packet field except DATA is a single character.  Each field
except START is composed only of printable ASCII characters.  The packet is
normally terminated by a carriage return.

START is usually Control-A (SOH), but can be redefined.
LEN   is the packet length, 0-94, encoded as a printable ASCII character.
SEQ   is the packet sequence number, 0-63 (recycles), printable.
TYPE  is the packet type, S F D Z B Y N, etc.
DATA  is a file name, file data, etc, depending on TYPE, printable ASCII.
CHECK is an 8-bit checksum, folded into 6 bits as a printable character.
```

Figure 3: Xmodem and Kermit Packets

The differences between Xmodem and Kermit are worth noting. First, Xmodem uses 8-bit binary bytes in its packet fields, and therefore requires an 8-bit transparent communication link. It cannot function, even for text files, when parity is in use. Similarly, when any device in the communication path is sensitive to control characters such as Control-Z or Control-S (which occur in the Xmodem packet control fields), Xmodem packets are subject to interference. For this reason, Xmodem cannot operate in conjunction with XON/XOFF or other in-band flow control. Kermit, on the other hand, encodes its packets as lines of text, and therefore does not have these restrictions.

Second, Xmodem packets are sent only in one direction. The responses are bare unchecked control characters such as Control-F for acknowledgement, Control-U for negative acknowledgement, or Control-X for cancel. Corruption of Xmodem responses into other valid responses is possible, and can cause a file transfer to terminate prematurely. Kermit uses fully error-checked packets in both directions, and is therefore more robust in the face of transmission errors.

Third, Xmodem uses fixed-length packets. There is no length field. If a file's length is not an exact multiple of 128 bytes, then extra bytes will be transmitted. Furthermore, if a computer, multiplexer, or other device cannot handle bursts of 132 characters, Xmodem packets will not get through. Kermit packets include a length field. Packets can be adjusted to accommodate small buffers, and a short packet can be sent at the end, so there is no confusion about the exact end of file.

Fourth, Xmodem includes no mechanism for transmitting the file's name, and therefore has no way of sending multiple files in a single session. Kermit does this routinely.

Fifth, Xmodem makes no distinction between text and binary files. But since the conventions for representing text files on different systems can vary, the results of an Xmodem text-file transfer between unlike systems can be surprising. Kermit specifies a common intermediate representation for text files during transmission, so that incoming text files can always be stored in a useful form. However, this places the burden on the user to select text or binary transfer mode.

Finally, both the Xmodem and Kermit protocols have seen a number of extensions over the years. Xmodem has no formal or consistent way to negotiate the presence or absence of given features, whereas feature negotiation is built into the basic Kermit protocol. A pair of variant Xmodem programs will not necessarily be able to communicate, whereas any pair of Kermit programs will automatically fall back to the greatest common set of options. Xmodem and Kermit protocol extensions include:

- Multiple files. MODEM7 and YMODEM can transfer multiple files in a single batch, Xmodem can't. Multiple file transmission is built into the basic Kermit protocol.
- The ability to pass 8-bit data through a 7-bit channel. Xmodem can't. Kermit supplies this as a negotiated feature (commonly available).
- Alternate checksums. Xmodem-CRC uses a 16-bit cyclic redundancy check for greater reliability, and tries to adapt itself to 8-bit-checksum-only Xmodem programs automatically. Kermit supplies an optional 12-bit checksum, and a 16-bit CRC, negotiated with automatic fallback to the single character checksum.
- File transfer interruption. Both Xmodem and Kermit allow file transfer to be interrupted cleanly. Kermit also includes the ability to cancel the current file in a group and proceed to the next one.
- Compression. Kermit programs may negotiate compression of repeated bytes. Xmodem lacks a compression option.
- Long packets. YMODEM allows 1K-byte fixed-length packets for greater efficiency. Kermit extensions permit variable-length packets up to about 9K, negotiated with automatic fallback to regular-length packets.
- Sliding windows. Kermit programs may negotiate simultaneous and continuous transmission of packets and their acknowledgments on full-duplex links, with a window of up to 31 unacknowledged packets, and selective retransmission of lost or damaged packets. (This option is not yet widespread among Kermit implementations). Sliding windows are not possible in Xmodem because its responses carry no sequence number (an Xmodem variant called WMODEM simulates sliding windows, but only works if there are no errors).
- File attributes. YMODEM transmits a file's name, size, and creation date. Xmodem does not. Kermit always transmits the name, and the ability to communicate a wide range of other file attributes may be negotiated (but, like sliding windows, this is not yet a widely implemented Kermit feature).
- Checkpoint/restart. ZMODEM includes the ability to restart a file transfer after the connection was broken. Neither Xmodem nor Kermit have this ability.

Kermit also differs from Xmodem by including a "file server" mode of operation, in which the remote Kermit program receives all its instructions from the PC Kermit in packet form. This simplifies operation considerably. Kermit servers can transfer files, as well as perform a variety of file management functions -- deletion, directory listing, changing directories, etc.

Implementations of Kermit can be had for most PCs, minis and mainframes. Xmodem implementations are found mostly on PCs, rarely on minis and mainframes. Basic Xmodem is somewhat more efficient than basic Kermit, because the packets are slightly longer and there is less encoding overhead. The situation is reversed when Kermit can do compression, long packets, or sliding windows.

Most commercial RS-232 communication packages claim to include Xmodem, Kermit, or both. In general, the commercial Xmodem implementations include none of the MODEM7, YMODEM, or ZMODEM options, but often do include support for CRCs. Thus, they can transfer only a single file at a time, and only through transparent 8-bit communication channels. The commercial Kermit implementations vary from the bare-bones to the very advanced, but all can transfer text files through 7-bit links, and can handle multiple files in a single operation. It is not always apparent from vendor literature exactly which options are supported, so if any of these issues are important to you, you should call the vendor and ask about them. After all, one of the advantages of commercial offerings over public domain software is telephone support.

OTHER ASYNCHRONOUS PROTOCOLS

Xmodem and Kermit are not the only two asynchronous communication protocols in the marketplace. Others include UUCP, Blast, MNP, X.PC, Poly-Xfr, DX, Compuserve, FAST, and DART. Most of these protocols are proprietary, which means that the protocol specification itself is secret, or licensed, and they are found primarily in commercial packages. They often include advanced capabilities like checkpoint/restart, bidirectional file transfer, and sliding windows.

But all proprietary protocols have the same drawbacks: you must buy commercial packages in order to use them, and if there is not a package available for a certain computer that you need it for, you're out of luck. Of the commercial packages, Blast probably comes closest to Kermit in covering a wide variety of systems, and exceeds Kermit in many design and performance areas. The drawback is the cost: $250 for the PC version, $450 for a PDP-11 version, and more for larger minis or mainframes. And since the Blast protocol is inherently full duplex, a special "Blast box" front end must be purchased for half duplex systems. Kermit, on the other hand, may be used with either full or half duplex systems, and the cost is minimal.

SUMMARY

Here is a checkoff list that you can use to evaluate and compare communication packages. Before purchase, you should decide which features are important to you, and then determine which packages have these features. Check the vendor literature, or call the vendor directly.

CONFIGURATION

Make and model of your computer:____

Operating system and version:____

Memory:___________(K) Floppy drives:_________ Hard Disk Capacity: _______(M)

Communications interfaces:____

Modem make and model:____________________________ [ ] Internal [ ] External

Name of communications package:____

Communications package vendor:_________________________ Phone:________________

Package memory size:___________(K) Package disk occupancy:________________(K)

Before proceeding, be sure that the communication package is compatible with your computer's configuration!

COST

(a) What is the unit cost of the package? $______________

(b) Is source code included, so that you can make changes and fix bugs? Is there is an additional charge for source code? Cost of source code, if you want it: $______________

(c) Is copying allowed? If so, go directly to (f).

(d) How many PCs will you need it for? _____________
Is there a volume discount? If so, enter discounted cost: $______________

(e) If a site license is available, what does it cost? $______________

(f) Enter best total price for PC versions . . . . $______________

(g) Do you also need minicomputer or mainframe versions?
If so, enter total cost for mini or mainframe versions . . $______________
(Figured as above)

(h) Total cost to your organization . . . . $______________

DOCUMENTATION, TRAINING, AND SUPPORT

Is the manual...
[ ] thick and unmanagable?
[ ] thin and cryptic?
[ ] just right?

How important is the manual?
[ ] Must be consulted frequently
[ ] Occasional lookups required

[ ] Does the manual have a good index and table of contents?

[ ] Is training available?

[ ] Is training necessary?

[ ] Is telephone support available and included in the package price?

WHAT IS YOUR PRIMARY USE FOR THE PACKAGE?

[ ] Long interactive remote sessions. Communication parameter settings, terminal emulation, and key definition are the most important features.

[ ] Infrequent remote sessions mainly for the purpose of data transfer. Concentrate on the user interface, script language, and file transfer protocol.

COMMUNICATIONS PACKAGE FEATURES

Each of the features listed below should be evaluated according to your needs. The lack of a certain feature is not critical if you know you will never need that feature. Items may be rated as follows:

X - I don't care about this feature.
Y - I need this feature, and the package has it.
N - I need this feature, but the package doesn't have it.

A single "N" may be sufficient to disqualify the package, depending on how important the feature is to you. If you don't know whether the package provides a feature you need, call the vendor and ask.

USER INTERFACE

[ ] Is help available at all times?

[ ] Does the user interface favor the novice user? (Menus at all times)

[ ] Does it favor the expert user? (No menus)

[ ] Is the package equally convenient for both novice and expert? (Menu on demand)

[ ] Can canned procedures be set up for unskilled users? (Scripts, command files)

[ ] Can local operating system functions be accessed without leaving the package?

[ ] Can the package be used by the disabled?

COMMUNICATION PARAMETER SETTINGS (always important)

Bits/Second: 0,110,300,1200,2400,4800,9600,19200,etc. Maximum:__________

Duplex
[ ] Full (e.g. for DEC minis)
[ ] Half (e.g. for IBM mainframes)

Echo
[ ] Remote (e.g. for DEC minis)
[ ] Local (e.g. for IBM mainframe linemode connections)

Data Bits
[ ] 5 (Baudot)
[ ] 7 (ASCII)
[ ] 8 (national characters)

Stop Bits
[ ] 1   (for most connections)
[ ] 1.5 (rarely used)
[ ] 2   (used only for 110 bits per second or less)

Parity Selection
[ ] None (all bits used for data)
[ ] Even (required by some mainframes, front ends, public networks, etc)
[ ] Odd (ditto)
[ ] Mark (ditto)
[ ] Space (rarely used, but sometimes handy)

Character Set Selection
[ ] 5-bit Baudot (used in Telecommunication Devices for the Deaf)
[ ] 7-bit US ASCII (most common in English-speaking countries)
[ ] 7-bit "national ASCII" (Norwegian, German, etc)
[ ] 8-bit "extended ASCII" (e.g. use of IBM PC 8-bit character set)
[ ] Support for international standard non-Roman character sets
[ ] User-definable or downloadable character sets

Flow Control Selection
[ ] X-on/X-off (e.g. with DEC computers)
[ ] ENQ/ACK (e.g. with Hewlett-Packard computers)
[ ] RTS/CTS (for half duplex modems)
[ ] Half duplex line turnaround handshake (e.g. with IBM mainframes)
[ ] Other:________________________________
[ ] None (can flow control be turned off?)

Debugging
[ ] Special display of all received and transmitted characters
[ ] Logging of all received and transmitted characters

[ ] Can you collect communication settings into recallable configurations?

CONNECTION ESTABLISHMENT

Support for RS-232-C asynchronous modem signals (RTS, CTS, DSR, CD, DTR, RI):

```
[  ] Does the package monitor Carrier Detect (CD) and Data Set Ready (DSR)
       from the modem?
[  ] Does the package assert Data Terminal Ready (DTR)?
[  ] Can the package drop DTR to hang up the phone?
[  ] Does the package respond to Ring Indicator (RI) so that it can be called
       from outside?
[  ] If you have a half duplex modem, does the package support RTS/CTS?

[  ] Does your PC have an internal modem?
[  ] Does the package support this internal modem?
```

Dialer Control:

```
[  ] Does your modem provide automatic dialing?
[  ] What dialing language is used by your modem? ________________________
[  ] Does the package support automatic dialing?
[  ] Does the package support your modem's dialing language?
[  ] Does the package provide a phone directory?

[  ] Can the package operate over direct connections, without modems?  That
     is, can it be told to ignore CD and DSR?  (If not, you will need the
     "fakeout" (minimal) null-modem cable from Figure 1).
```

Script language for automatic login, unattended operation:

```
[  ] Access to all necessary package commands from script language.
[  ] Conditional execution/termination of script commands.
[  ] Fancy script programming features (variables, labels, goto's, etc.)
[  ] Unattended operation (e.g. late at night, when phone rates are low).

[  ] Can the program be suspended and resumed without dropping the connection?
```

TERMINAL EMULATION:

What terminal(s) does the package emulate? ______________________________

[ ] Is the maximum speed for full duplex terminal emulation sufficient for your needs?

[ ] Does the package emulate a terminal that is supported by the computers you wish to communicate with?

[ ] Is the terminal emulated fully enough for use with all desired software applications on these computers?

[ ] Is any special hardware (like a 132-column board) required in the PC?

[ ] Does the package support fore- and background colors? (Do you need them?)

[ ] If a graphics terminal is emulated, does your application support it?

[ ] Screen rollback (view screens that have scrolled away)

[ ] Screen dump (save current or previous screens in PC files)

[ ] Printer control (copy displayed characters to printer;print whole screen)

[ ] Print or save text screens in alternate character sets

[ ] Print or save graphics screens

[ ] Function keys

[ ] Key redefinition

[ ] Keystroke macros

[ ] Translation of displayed characters, alternate character sets

---

FILE TRANSFER PROTOCOLS

[ ] ASCII (this is not an error-correcting protocol)

[ ] XON/XOFF (this is not an error-correcting protocol)

[ ] Xmodem

[ ] Kermit

[ ] Proprietary (Blast, MNP, etc):______________________________

[ ] Other: ______________________________

[ ] Do the systems you’re communicating with support the same protocol(s)?

[ ] Does the package transfer both text and binary files?

[ ] Do text files arrive on the target computer in useful form?

XMODEM OPTIONAL FEATURES

[ ] Modem7-style transfer of multiple files

[ ] Xmodem-CRC for more reliable error checking

[ ] Ymodem 1K packets for increased efficiency (half duplex)

[ ] Ymodem filename transmission

[ ] Checkpoint/restart (Zmodem)

[ ] Wmodem continous transmission (full duplex)

[ ] Do the computers you wish to communicate with support the same Xmodem options?

KERMIT OPTIONAL FEATURES

[ ] 8-bit data through 7-bit links (e.g. links with parity)

[ ] Repeated character compression for improved efficiency

[ ] 12-bit checksum, 16-bit CRC, for more reliable error checking

[ ] File transfer interruption

[ ] Long packets (up to 9K) for improved efficiency (half duplex)

[ ] Sliding windows for improved efficiency (full duplex)

[ ] Transmission of file attributes

[ ] Server operation

[ ] Remote host commands and file management