Special Report: A Closer Look at the Tandy 2000

80micro

May 1984 USA $4.00

A WAYNE GREEN PUBLICATION

the magazine for TRS-80* users

Be the Boss Of a BBS

Plus:

- Equation Occasions
- Your III Becomes a 4
- A New Pascal Column
- VisiCalc Enhancements

m: Pete S. 031050.34

: John C. 120947.3a

John.

Now that you've got the BBS up and r

probably swamped with mail, bu

Jeanne for your hospitality wh

0 74470 65947 05

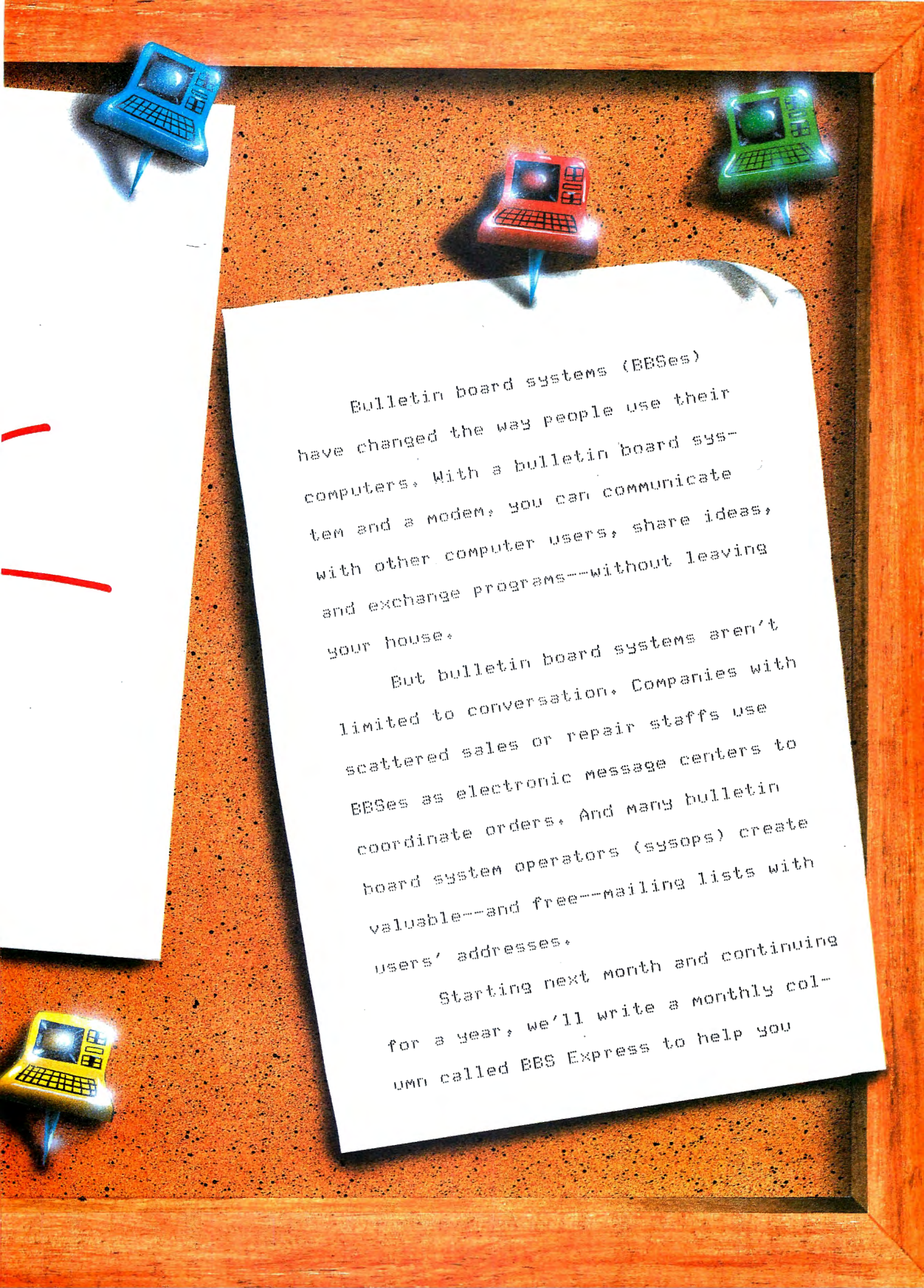
80micro 5/84 #52 A

BBS Express

by J. Stewart Schneider and Charles E. Bowen

Become a modem mogul and a BBS boss!

As a bulletin board system operator, you can run your own communications network.



Bulletin board systems (BBSes) have changed the way people use their computers. With a bulletin board system and a modem, you can communicate with other computer users, share ideas, and exchange programs—without leaving your house.

But bulletin board systems aren't limited to conversation. Companies with scattered sales or repair staffs use BBSes as electronic message centers to coordinate orders. And many bulletin board system operators (sysops) create valuable—and free—mailing lists with users' addresses.

Starting next month and continuing for a year, we'll write a monthly column called BBS Express to help you

create and run a bulletin board system with your 48K two-drive Model I, III, or 4. (Alternatively, you can order the finished program from us [see the sidebar].) As a preface, this month we'll discuss the theory behind computer communications.

The notion of becoming a SYSOP is appealing, but it's not as simple as it looks. Before you can communicate with your TRS-80, you must overcome some serious problems involving telephone communications and your computer's design.

Design Problems

First of all, your TRS-80 has a memory-mapped video screen and a memory-mapped keyboard. For every screen location, a memory location stores a byte representing the character displayed there. And for every key on the keyboard, the computer assigns a tiny switch to a specific memory location that it scans to determine whether or not you're pressing the key.

When you operate a TRS-80 over the phone, your commands go out the communications line. PRINT@ statements become useless because you have no print destinations. And programs that produce screen displays by POKEing ASCII values into screen memory won't work.

Your computer also receives input from the communications line. As a result, games that scan the keyboard with PEEKs and word processors that scan switches hooked up to the keys won't work.

Most microcomputers use the video screen as the default output device and the keyboard as the default input device. TRS-80s are designed to function primarily with immediate or self-contained peripherals, and contain the coding necessary to use only them.

On the other hand, you can scatter minis and mainframes throughout a building or across a continent. A terminal in Tallahassee can access the main computer in Minneapolis. And the main computer can communicate with the disk drive in the next room or print a message in Pittsburgh.

On a large computer, the central processor spends lots of time routing messages and controlling access to peripherals. Personal computers can't perform such complicated message switching. If you want to run a personal computer from a distance, you're asking for trouble.

Telephone technology only compounds the problem. For example, when your computer sends a character over telephone lines, the telephone company does some business with it and delivers it to a distant computer. That computer does some business with it, and echos it back over the lines to your computer. None of this happens instantaneously, and the two machines quickly get out of synchronization.

Parallel transmission requires eight wires to work. The first problem is that the telephone company doesn't supply ribbon cable on its telephone poles.

The Data

To overcome these problems, you need to understand what you're transmitting and what it looks like. Some of you might find the next few paragraphs elementary; think of them as a review. Lots of new material follows.

Computers speak a confusing language of ones and zeros called binary. Single ones and zeros are called bits, and eight of them make up a byte. Your computer works by manipulating these bytes, with each one representing a letter, punctuation mark, part of a number, or instruction.

A byte can also be part of a program, but once you remove it from the originating computer and put it on the telephone line, the receiving computer can't tell what it represented.

Let's make that clearer. Suppose you somehow connect two TRS-80s to the telephone line, and send the number 191 from one to the other. The receiving computer will get a 191 if you do it right, but what does that 191 mean? It can be a graphics character, a Z80 instruction, or part of a Basic program. Once the transmitting computer sends a byte over telephone lines, the receiver can't tell what the byte means—unless the two computers have reached some agreement beforehand.

You could decide to send and receive only the letters of the alphabet, and certain punctuation marks, numerals, and control characters. Since computers use binary numbers, you could assign a number to each letter, digit, and punctuation mark. You'd also need characters for control, such as carriage returns and line feeds.

After you decide which numbers represent what, you could make a table of the number/letter combinations and mail it to the person with whom you'll be communicating. Then he could program his computer to translate the numbers you send into the letters and digits you intended.

Fortunately, you don't have to go to that trouble; someone's done the work for you. The list you need is the American Standard Code for Information Interchange (ASCII) set (see your computer's operating manual for the complete list of ASCII values).

Parallel and Serial

To understand how you transmit and receive ASCII-encoded data, we need to discuss the hardware involved. Inside your TRS-80, bytes of data zip around on the data bus, which consists of eight wires similar to those of a printer cable. Each bit in a byte travels in its own wire on the data bus and goes from here to there with other bits, like an ocean wave approaching the beach.

This is called parallel transmission, and it requires eight wires to work. The first problem is that the telephone company doesn't supply ribbon cable on its telephone poles. To transmit a byte of data, you need to fit it into two wires, the two the telephone uses.

A byte of data flows along the data bus with some of the wires containing a one, some a zero. Together they represent a binary number, such as 11011011.

Suppose you could send the contents of each wire over telephone lines. Since each bit is off (zero) or on (one), you

The Key Box

Models I, III, and 4
48K RAM
Disk Drive

can send the bits one at a time, in serial transmission.

A device called the universal asynchronous receiver/transmitter (UART) makes serial transmission possible. UARTs are asynchronous because the data they handle does not carry a clock signal to keep the two communicating computers synchronized. UARTs are the main working components of RS-232 boards.

A transmitting UART receives a byte of data from the data bus, collects the bits, and sends them one by one to the receiving UART. The receiving UART collects the bits from the communications line and reassembles them into a byte.

Simply having a UART in your TRS-80 doesn't mean you can communicate, however. You need a terminal program that tells the computer to access the UART.

Intelligent Software

A terminal program catches your keystrokes and sends them to the UART, then catches the input from the UART and sends it to the screen. A dumb terminal program does only this. Communications software that performs other functions, such as storing incoming messages, automatically dialing the telephone, or printing messages, is what's known as an intelligent terminal program.

Once you have two computers equipped with UARTs and terminal programs, you can connect them together by their UARTs and communicate between them.

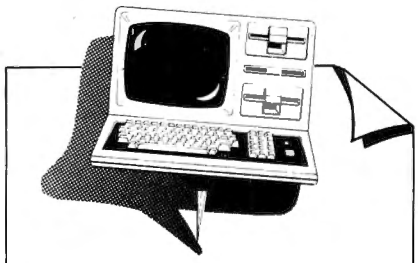
But the telephone network is designed to send sounds, not the digital data your computer produces. Converting your ons and offs into sounds means more hardware.

First, you should decide on a scheme of tones, say a beep for a one and a boop for a zero. Then you could code your data into a series of beeps and boops, send that out, and translate it back on the other end.

You need a device that receives the ones and zeros from the UART, modulates them into tones, then sends the tones over the long distance lines. You also need a device that receives the tones, demodulates them into ones and zeros, and feeds them to the UART for reassembly into a byte. These modulators/demodulators are known as modems.

One small problem remains. The local computer beeps. The distant computer beeps. Which beep came from what computer? Confusion results if both the distant and the local computers use the same tone to represent ones and zeros. You should give the computers different sets of tones to use so each modem will know which computer beeped. We'll call one set of tones the answer set and the other the originate set.

These terms are misleading; separate tones only let the modems tell which computer made which sound. It doesn't



The local computer beeps. The distant computer beeps. Which beep came from which computer? Confusion results if both computers use the same tone.

matter which computer originated the call and which answered the telephone, only that one modem is in the originate mode and the other is in the answer mode.

You're almost ready to begin communicating. The terminal program scans the keys and sends the keystrokes to the UART. The UART collects bits and sends them to the modem. The modem beeps and boops to the telephone line. On the other end, nothing predictable happens. Why?

Transmission Conventions

You haven't yet decided on a convention concerning when or how fast to transmit.

The bits flying along on the data bus inside your computer stay together because a system clock keeps them synchronized. Two computers in a telecommunications set-up might be separated by a continent, two satellite links, and a hand-crank switchboard. And each computer has a unique system clock.

The receiving computer can't begin to understand the bits pumped out by the transmitter unless both use a consistent format for transmission.

To resolve this, your computer uses start and stop bits to mark the beginning and end of a data byte. Before the computer sends a data bit, it sends a start bit to get the receiving UART's attention. Then it sends the data bits, followed by 1 or 2 stop bits.

When the receiving computer sees the start bit, it knows a character is coming and accepts the data bits. Finally, it accepts the stop bits.

If both computers agree that each byte will contain 8 bits, stop and start bits seem superfluous. After all, any TRS-80 can count to eight.

But when you send and receive high-speed serial data, it's possible that a bit will get lost along the way. In fact, it's a certainty. And when your line drops one or more bits, the receiving computer's counting schedule gets upset. Without the stop bit, it would miss the error and be out of synchronization from then on.

The UART in your TRS-80 can handle word lengths of 5, 6, 7, or 8 bits. Of course, both transmitter and receiver must agree on the number of bits between start and stop bits.

The ASCII set requires at least 7 bits to a word, because the largest ASCII number is 127, which requires 7 binary bits.

When a transmitting computer isn't sending information, it creates a continuous beep, or a logic one. The first boop it sends is the start bit. The transition between a beep (a logic one) and a boop (a logic zero) indicates that a byte of data will follow. If a computer sends two or more ones or zeros together, they create a steady tone.

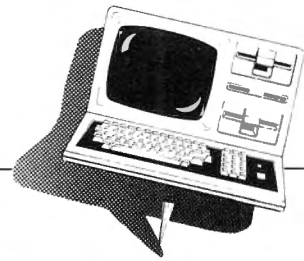
Perfect Timing

Your computer uses a method called bit time to regulate the frequency of bit transmission. Both UARTs must use the same bit time.

For example, with a bit time of .003333 seconds, both computers transmit a bit every .003333 seconds. If a computer hears a beep for the first .003333 seconds, the bit is a one. If it hears a boop, the bit is a zero.

It's awkward to describe a UART's setting by bit time with all those zeros and threes. The setting is commonly described by the reciprocal of the bit time, known as the baud rate.

The reciprocal of a number is one divided by that number. One divided by .003333 is approximately 300; the baud



Express Speca

Building a bulletin board system is a formidable project. During this series you'll work with every part of your TRS-80 but the power cord. When it's over you'll have a solid grasp of telecommunications and a nifty piece of software for your 48K, two-drive Model I, III, or 4.

Every month, BBS Express will include a program listing you should enter and append to the previous month's work. If you already have a modem and terminal program, you can visit our BBS and download the ASCII files for each month's listing.

If you can't wait to see your bulletin board in action, we'll send you a working version. Send a formatted disk and $10, or no disk and $15, to Saturday Software, P.O. Box 404, Catlettsburg, KY 41129, and specify whether you want the TRSDOS or LDOS version.

You can see the bulletin board in action by calling 606-739-6088 after 6 p.m. eastern standard time. We hope you'll give us a call—the board has a special section for your comments, questions, and suggestions.

Our BBS has message and data-base space, plus separate sections with limited access for private messages. Its command structure is similar to CompuServe's, and includes scanning, dynamic screen formatting, and optional line feeds after carriage returns.

Most of the board uses Basic, but time-sensitive parts, such as the receiver section, use machine code.

Since strangers of sometimes impure motives will be managing your BBS files, the system is crash-proof, reliable enough to take care of itself when left alone, and fast.

We're looking forward to communicating with you in the coming year. □

rate in the above example is 300. A bit time of .00083333 equals a baud rate of 1,200.

You should not confuse baud rate with bits per second (BPS). BPS is the number of data bits transmitted in one second. Baud rate, on the other hand, represents the number of all bits—stop, start, and data—transmitted or received in a second.

Error Terror

A major part of any telecommunications system is devoted to detecting and correcting errors. Nationwide, telephone lines experience 1.5 bit errors for every 100,000 bits transmitted. That's about one error every four minutes of continuous transmission at 300 bits per second.

A discussion of all the possible plans for error-checking is beyond the scope of this article, but we'll examine two of the most popular methods.

UARTs use a method known as parity, adding a single bit to the end of every byte. To see how it works, let's examine the byte of binary data discussed earlier, 11011011.

This byte contains six 1 bits, an even number. In an even parity setting, the UART would add a zero bit before transmission, keeping the total of 1 bits an even number. In odd parity, it would add a 1 bit and make the number of 1 bits an odd number. Discrepancies from the parity setting, when a byte with an odd number of 1 bits shows up in even parity or one with an even number of 1 bits turns up in odd parity, alert the computer to transmission errors.

Computers can also ignore parity, using a no parity setting.

Another popular method of error detection, duplex, involves how your typing gets to the screen. Typing for transmission to a distant computer is like working in the dark—you can't see what you're doing. Half-duplex makes the terminal program echo the letter represented by your keystroke to your screen. But half-duplex provides no error-checking.

A full-duplex system sends your keystroke to the distant computer, which sends it back for display. Full-duplex requires your modem to receive and transmit at the same time. If you type an A, and a B appears on the screen, you can backspace and correct it. Errors can occur during transmission or echo, but you should assume they happen during transmission.

Proper Protocol

Your computer is pumping out

thousands of bits every second. On the other end, the receiver is catching the bits. What happens when the receiver's disk drive comes on? Or when the receiver has to do something with the received data, such as scroll the screen? Your computer goes on transmitting merrily, the receiver stops listening, and the data gets lost.

We've come a long way, but you still need a method for the two computers to tell each other, "Wait a minute, I'm busy."

The most common method is XON/XOFF handshaking protocol. When the receiver must stop listening, it sends an XOFF control character to the transmitter, which does one of two things.

The transmitter either stops transmitting until it receives an XON, instructing it to resume transmitting, or ends the connection when enough time has passed since the last echo for it to assume the connection is broken.

XOFF is defined in the ASCII set as DC3, device control 3. It is 13 hexadecimal (hex) (19 decimal); press control-S to enter DC3. XON is DC1 or 11 hex (17 decimal). Press control-Q to enter XON. CompuServe users know these controls as the freeze and unfreeze commands.

We tend to think of electricity traveling instantly from place to place, and from our reference point, it does. But computers work in a more precise time frame, and the time it takes for information to travel over telephone lines quickly confuses XON and XOFF commands.

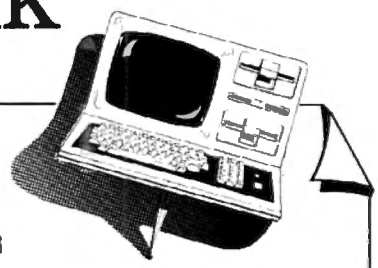
For example, the receiver stops receiving to perform some action and sends an XOFF to the transmitter. But characters are still headed for the receiver, because it takes time for the XOFF message to reach the transmitter and more time for the transmitter to react.

The receiver must anticipate its needs and send the XOFF message before it needs to stop. And since TRS-80s don't have the code to do this, the terminal program has to handle it.

That's about it. Connecting two TRS-80s over phone lines is more complicated than it first appears, but you can do it if you're willing. After communicating with your computer, playing games and balancing your checkbook will never be the same. ■

Contact J. Stewart Schneider and Charles E. Bowen at Saturday Software, P.O. Box 404, Catlettsburg, KY 41129.

Teaching Your TRS-80 to Talk



An electronic bulletin board works like the bulletin board at your local supermarket, but sending messages on a small computer is a lot more complicated than tacking them up at the corner grocery.

Last month we introduced you to electronic bulletin boards and described how computers communicate with one another over telephone lines (see "BBS Express," May 1984, p. 42). But there's more involved in bulletin boards than teaching your TRS-80 to talk on the telephone. Over the course of this series we'll take a look at telecommunication theory, disk file formats, Z80 Assembly language, and Basic programming tricks. You'll learn to write a complex program and modify an existing operating system. You don't have to be an Assembly-language programmer to follow the discussion, but it would help if you're familiar with editor/assemblers.

Each month we'll present another module of the BBS and explain how each module fits into the finished bulletin board and justify our programming decisions. The code for each module is available on Load 80, and as we mentioned last month, you can download the code from our BBS (606-739-6088) after 6 p.m. Eastern Standard Time.

Impatient readers can order a working version of the BBS from us (see ordering information at the end of this article).

Switching Messages

An electronic bulletin board is essentially a telecommunicating message-switching program. It works like the bulletin board at your local supermarket, but sending messages on a small computer is a lot more complicated than tacking them up at the corner grocery.

A visitor to your BBS wants to send and receive information, and you must guide him through the system. Simply linking your TRS-80 to the telephone line with a modem isn't enough. A good BBS should prompt the user, provide a method for sending and receiving information and a directory of available information, and be absolutely crash-proof.

A BBS is an operating environment for your computer that limits the user's choices, provides input/output facilities, monitors the telephone connection, and handles files.

Bulletin board systems are most popular at night, when long-distance telephone rates are lowest. Since you're probably not going to be there if someone uses the BBS at 3 a.m., your BBS software should be helpful enough to offset its lack of documentation.

Environmental Conditions

Before writing the BBS, you must decide on a system and an environment, list the system's functions, and make Basic talk.

Choosing an environment for the BBS Express involves some arbitrary decisions. We chose a Model III with 48K RAM, two 40-track disk drives, a Daisy Wheel II printer, and a Hayes Smartmodem. (The system does not require the printer.) For operating systems, we chose TRSDOS 1.3 and LDOS 5.1.3.

We can already hear Models I and 4 owners grumbling. Let us explain our decisions. The BBS is not a small program—it requires about 28K to run. Although you could run it (with alterations) as a series of modules on a machine with less memory, the BBS would run so poorly that it wouldn't be worth the effort. To work properly, the BBS Express needs a 48K machine.

Model 4's pose a different problem. Running in Model 4 mode, the computer switches out its ROMs. Since our BBS uses ROM calls, we can only support Model 4's operating in Model III mode.

Our DOS choices will probably create a lot of flak, too. We want to examine different operating systems in the BBS Express, and investigate the problems inherent to designing a system that runs in different environments. We chose TRSDOS 1.3 because Model IIIs came with a copy of it, and LDOS 5.1.3 because it approaches the same problems in a radically different way. This doesn't mean the BBS won't run on your operating system.

Since the machine-code portions of the program don't interact with the DOS, and the remainder of the program is in standard Basic, the BBS Express might run very well with your DOS. We hope so. Try it and let us know how it goes.

Parameter Choices

We also had to decide on communications parameters, often inaccurately

referred to as communications protocols. Last month we discussed baud rate (the speed of information transfer), word length (the number of bits in a word), stop bits (the number of bits signifying the end of a character), and parity (a method for error-checking).

To set these parameters, you must configure your universal asynchronous receiver/transmitter (UART). Your baud rate depends on the rates your modem supports. You can't set your UART for a 1,200 baud rate if the modem only supports 300 baud, the most popular rate for small computers. Our Hayes Smartmodem is a 300 baud unit.

Word length, you will recall from last month, can be 5, 6, 7, or 8 bits. Many BBSes use 8-bit words, but you can send the entire ASCII set using only 7 bits. CompuServe Information Service uses 7-bit words, and our BBS also uses 7-bit words. In fact, we designed the BBS Express to work as much like CompuServe as possible. We followed CompuServe protocol the rest of the way, using even parity and 1 stop bit.

Specifications

So far, we've picked a machine, two operating systems, and communications parameters. We have no code or any idea of what's going to happen when somebody calls in, but at least we're approaching the project in a systematic manner.

We need to decide on the operational specifications for the program. After a little scratching with a pencil and paper, we came up with a program Wish List (see the Figure). It's not a formal design specification, nor does it cover all the areas we must address in the BBS Express. In later installments, we'll expand our list.

Our Wish List contains some pretty exotic terms. Don't worry if they look strange; demystifying them is what the BBS Express is all about.

The specifications in the list should give you an idea of why we chose the Model III for the BBS. One hundred 20-line messages take up 128,000 bytes, and a Model I has only 89,000 bytes available on a data disk.

Talking Basic

To create a BBS, you must teach Basic to talk, and to do that, you must learn how to communicate with the UART.

Model IIIs and 4's have port-mapped serial input/output (I/O). You communicate with the UART through its ports, rather than by PEEKing and POKEing in memory. But working with ports is similar to working with memory—easier, in fact.

The Z80 chip that's the brain of your TRS-80 can communicate with 255 different ports. The computer performs operations to the port you specify. Basic has two commands that can access the ports, INP and OUT port,value. INP reads from a port and OUT writes to a port. In Z80 Assembly language, the two commands are IN A,(port) and OUT(port),A.

1. Send and receive information from the telephone lines.
   A. Information must be readily available to Basic.
   B. Must detect broken connection and bring the program to normal termination in that case.
2. Maintain a message board of at least 100 20-line messages.
   A. Message board must be readable forward, backwards, individually, selectively, or by marked messages.
   B. Message board must take care of file clean-up, removing files that have been deleted, or that have scrolled off, and must be indexed for quick operation.
   C. Must support printer so that SYSOP can have hard copy of messages.
   D. Users must have a way of editing messages before storing them.
   E. Must provide a method for marking messages directed to a user, and informing him that he has messages waiting.
   F. The message board should be partitioned into sections, and SYSOP must have control over who has access to what sections.
   G. Private messages must be allowed.
3. Allow for a large database.
   A. Must include a directory or catalog of files available, preferably alphabetic.
   B. Must include provision for downloading program material without formatting to user's screen width.
   C. Password protection desirable for SYSOP's private use.
   D. Must react quickly enough to support burst-transmissions from another computer, and support XON/XOFF handshaking.
4. Maintain a membership log.
   A. Provision for production of mailing labels would be handy.
   B. Must have fast, easy location of individual member's record.
   C. Should produce alphabetic listing, and support printer.
   D. Must allow the user or the SYSOP to alter user's record conveniently.
5. Maintain a chronological user log.
6. Dynamically format the screen displays to the user's screen.

Figure. Preliminary program outline.

Port	Read Function	Write Function
E8 hex	Modem status	Master Reset
E9 hex	Sense switches	Select baud rate
EA hex	UART status	UART control
EB hex	Receive data	Transmit data

Table 1. UART port functions.

The UART communicates with the TRS-80 through four of these ports, E8–EB hexadecimal (hex); see Table 1. To make the UART work, you must read from or write to the port assigned a specific instruction or command. An individual port may have one function for reads and another for writes; the direction of data flow determines port operation.

You must read from and write to these ports to set the communications parameters. For successful communications, you have to specify baud rate, parity, word length, and the number of stop bits; set both computers to the same values. Port E9 hex is supposed to set baud rate, so let's start there.

You might try writing the actual baud rate out to the port to set it. Unfortunately, that won't work. The largest number you can write to or read from a port is 255, and your baud rate is 300. Instead, you must write a coded value to this port, and set it to transmit and receive. Though perhaps not practical, you can receive and transmit at different baud rates.

The baud rate port works in a strange way. You must send the code for receiving and the code for transmitting in 1 byte. The code for 300 baud is five. Since you want to send and receive at 300 baud, write a 55 hex byte to the port. The Basic command for this parameter is OUT&HE9,&H55. The Z80 Assembly language command is LD A,55H OUT (0E9H),A.

Hex values for code	Decimal values for baud rate
2	110
4	150
5	300
6	600
7	1,200
A	2,400
C	4,800
E	9,600

Table 2. Baud rate codes.

Before you can send either command, you must use Reset to get the UART into a known status by writing any value to port E8 hex.

Building Bytes

You can now set the rest of the parameters with the UART control port, EA hex. This port might seem confusing if you're unfamiliar with the concept it uses. Port EA hex is bit-mapped: each of its 8 available bits has a different meaning (see Table 3).

You're going to build a byte from the information in Table 3. The UART uses this byte as a map of the functions it performs. If you want a feature, you can set or reset the appropriate bit. To set a bit, make it a 1, and to reset it, make it a zero. Let's start with a clean slate, the binary number 00000000.

Table 3 indicates that position zero is data terminal ready. Many auto-answer modems require data terminal ready to be set, and Table 3 indicates that you'll need a zero bit in position zero, so your byte remains 00000000.

Bit 1 is request to send, a signal used in half-duplex transmissions and not needed here. You should set request to

Bit Position	Function	Commands
0	Data terminal ready	1 = reset, 0 = set DTR
1	Request to send	1 = reset, 0 = set RTS
2	Break	1 = normal, 0 = true
3	Parity enable	1 = disable, 1 = enable
4	Stop bit select	1 = 2 bits, 0 = 1 bit
5	Word length	1 = add 2 bits
6	Word length	1 = add 1 bit
7	Parity	1 = even, 0 = odd

Table 3. Bit map of port EA hex.

Don't worry if some of the terms look strange; demystifying them is what the BBS Express is all about.

send, keeping your byte 00000000.

According to Table 3, bit 2 sets modem break. You can use modem break to get a BBS user's attention or to break a program in some installations. You want normal operation, so you should use a 1 in bit 2. Your byte is now 00000100.

Bits 3 and 7 work together to determine parity. Bit 3 determines whether or not you enable parity, and bit 7 signifies even or odd parity. You should set a zero in position 3 to enable it, and a one in position 7 for even parity. Your byte is now 10000100.

Bits 5 and 6 determine word length. If you set both bits to zero, the word length defaults to 5. Setting bit 5 adds 2 bits to word length, for 7-bit words. Setting bit 6 adds 1 bit to word length, for 6-bit words. Setting both 5 and 6 adds three, for 8-bit words. You want 7-bit words, so set bit 5, making your byte 10100100.

That's what you send out to port EA hex. Since it's awkward dealing with all those zeros and ones, you can express it as decimal 164, or A4 hex. Writing this value to port EA hex sets your communications parameters to 7-bit words, even parity, and no stop bits. The Basic command for this is OUT &HEA,164 and the Assembly-language command is LD A,0A4H followed by OUT (0EAH),A.

Let's summarize. First, you reset the UART by writing any value to port E8 hex. Then you set the baud rate by writing 55 hex to port E9 hex. Finally, you set 7-bit words, 1 stop bit, and even parity, by writing 164 to port EA hex.

Two examples might clarify this operation. Listing 1 contains the Z80 Assembly code, and Listing 2 contains the equivalent Basic commands.

Here's the end of the second BBS Express installment, and we still haven't written any code. But we decided on a system and an operating environment, and examined some program features. Next month the programming starts. ■

For a working version of the BBS Express, send $10 and a disk, or $15 and no disk, to J. Stewart Schneider and Charles E. Bowen, Saturday Software, P.O. Box 404, Catlettsburg, KY 41129. Be sure to specify TRSDOS 1.3 or LDOS 5.1.3.

Program Listing 1. Assembly-language instructions to program UART.

```
00100       OUT   (0E8H),A       ;MASTER RESET
00110       LD    A,55H          ;300 BAUD
00120       OUT   (0E9H),A
00130       LD    A,0A4H         ;7/E/1
00140       OUT   (0EAH),A
00150       END
```

End

Program Listing 2. Basic instructions to program UART.

```
100 OUT &HE8,0    :REM   MASTER RESET
200 OUT &HE9,&H55:REM    300 BAUD
300 OUT &HEA,164 :REM    7/E/1
```

End

✓ See List of Advertisers on Page 227

Programming the Communications Module—The Heart of the BBS

You went to great lengths last month to configure the universal asynchronous receiver/transmitter (UART) from Assembly language and Basic. We must confess. You could've done the same thing more simply with LDOS or TRSDOS commands. But command or no command, you should understand how to communicate directly with the UART. It's particularly important with TRSDOS, because its SETCOM command is a cranky piece of software.

Now you're ready to program the BBS's communications module. You'll also examine device control blocks and learn how the CPU transfers data to and from the UART.

Since you might use these commands in other applications, we'll include them. The SETCOM command for our BBS is:

```
SETCOM(WORD=7,BAUD=300,STOP=1,
PARITY=2,WAIT)
```

Typing SETCOM without any parameters displays the UART's present state. You might have noticed that SETCOM provides a parameter we haven't discussed: Wait or No Wait. If you specify Wait, calls to the ROM serial routine won't return until the UART receives a character—a splendid infinite loop.

LDOS doesn't have a SETCOM command and it avoids the Wait parameter. Since LDOS is a device-dependent operating system, you configure the UART with the same command you use for any device. The LDOS command for our BBS is:

```
*CL to RS232T/DVR (B=300,W=7,S=1,
P=ON,EVEN)
```

The Model III also has three ROM calls that manipulate the UART: $RSINT, $RSRCV, and $RSTX. You won't be using them, but if you're interested, the manual explains these calls in detail.

UART Talk

You'll remember from last month that the UART communicates its status to the CPU through the Z80 ports. The CPU transfers data to and from the UART using the same Z80 ports you used to configure the UART. After doing some handshaking with the CPU, the UART communicates data through port EB hexadecimal (hex).

In the first installment of the BBS Express (May 1984, p. 42), we explained how the UART accepts a byte of information from the CPU, strips the bits off one by one, and sends them to the modem for transmission over the phone lines. On the other end, the UART receives the bits, reassembles them into a byte, and presents the byte to the CPU. All this stripping and reassembling takes time—the UART can't work as fast as the CPU.

To account for this, the UART uses status port EA hex to tell the CPU, "I'm busy." (The function of a port depends on the direction data travels through it. Sending a byte to port EA hex sets the communications parameters and reading a byte indicates UART status to the CPU.)

Table 1 should help you navigate the bit map of port EA hex. Notice that only bits 3–7 communicate information. Bits 3, 4, and 5 detect errors that the BBS can't use, and bits 6 and 7 constitute the UART's busy signal. A UART is busy if it has no character ready for the CPU to pick up or if the UART hasn't yet sent the last character.

Bit 6 indicates to the CPU that the UART hasn't sent the last character, and bit 7 indicates that data is available. In other words, the CPU must wait until bit 6 is a one to transmit data to the UART, and it must wait until bit 7 is a one to receive data.

Data travels to and from the UART

The Key Box LOAD 80

Models I, III, and 4
48K RAM
One Disk Drive

through port EB hex. If bit 6 in port EA hex is a one, the UART gets its data from port EB hex. If bit 7 in port EA hex is a one, the UART sends its data to port EB hex.

Now that you have some background, you can put it to work, starting with the keyboard.

Routine ROM

TRS-80s use a ROM routine to check the keyboard for pressed keys. Since your BBS needs to pick up characters from both the keyboard and the communications line, you want the CPU to scan the UART every time it scans the keyboard. With a direct link from the UART to the ROM's keyboard-scanning routine, the routine will pass characters from the UART to the CPU, and if the UART is busy, the CPU will scan the keyboard.

Once you link the ROM routine to the UART, a Basic routine such as INKEY$ or Input can pick up characters from the UART as easily as from the keyboard. All you need do is figure a way to get the key scanning routine in ROM to also check the UART.

This could be an unsolvable problem, because the code that scans the keyboard is burned into ROM—you can't change it. Fortunately, you can get around this by changing the device control blocks.

A device control block is a switchboard that tells the ROM routine where to look for the keyboard-scanning code. Since the keyboard-scanning code ends with a return instruction, consider it a machine-language subroutine.

We'll make that clearer. Before the operating system scans the keyboard, it executes some code in ROM that you can't change. That code does some business, then checks the device control block for the address of the code to scan the keyboard.

You need to get into the device control block and make its address point to your routine. To get back to where you started in ROM, you need only end your routine with a return instruction.

The device control block for the keyboard is located at 4015 hex in RAM, and you can change it. Relative byte zero in a device control block contains a code for the type of device it controls (see Table 2). Relative bytes +1 and +2 contain the address of the routine that handles that device.

The video's device control block, located at 401D hex, is laid out the same way as the keyboard's. Now you need to hitch the video to the communications line, so that material headed for printing on the BBS's screen also goes to the modem.

Video Echo

Your callers expect an echo to their video screens. Full-duplex communications require that the host (your BBS) echo received characters back to callers as a primitive error-check. Program Listings 1 and 2 are an attempt to solve this problem, and the first real code on this expedition into telecommunications programming. Listing 1 is the TRSDOS version and Listing 2 is the LDOS version.

The action starts in line 700, at label

Bit	Function
0	Unused
1	Unused
2	Unused
3	Parity error (1 = true)
4	Framing error (1 = true)
5	Overrun error (1 = true)
6	Transmitter buffer empty (1 = true)
7	Data received (1 = true)

Table 1. Bit map of port EA hex.

Relative Byte	Contents
+0	DCB type
+1	LSB driver address
+2	MSB driver address

Table 2. Relative bytes in a device control block.

Program Listing 1. Linking video, keyboard, and RS-232 in TRSDOS 1.3.

```
                00100 ;       TITLE   <LISTING 1 - TRSDOS>
                00110 ;       VIDEO, KEYBOARD AND RS232 LINKING
                00120 ;       TRSDOS 1.3
                00130 ;
                00140 ;
                00150 ;
401D            00160 VBLK    EQU     401DH           ;VIDEO DCB
4015            00170 KBLK    EQU     4015H           ;KB DCB
00EB            00180 UDATA   EQU     0EBH            ;UART DATA PORT
00EA            00190 USTAT   EQU     0EAH            ;UART STATUS REGISTER
00E8            00200 MSTAT   EQU     0E8H            ;MODEM STATUS PORT
0033            00210 PRINT   EQU     33H             ;ROM PRINT ROUTINE
002B            00220 KEYBRD  EQU     2BH             ;ROM KEYBOARD SCAN
0A9A            00230 BASIC   EQU     0A9AH           ;PASS TO BASIC
402D            00240 TODOS   EQU     402DH           ;JUMP TO DOS
                00250 ;
                00260 ;
                00270 ;
                00280 ;       NEW VIDEO DRIVER PATCH
                00290 ;
0000 F5         00300 VIDEO   PUSH    AF              ;SAVE FLAGS
0001 C5         00310         PUSH    BC              ;SAVE CHARACTER
0002 DBEA       00320 VWAIT   IN      A,(USTAT)       ;CHECK UART FOR CLEAR
0004 E640       00330         AND     40H             ;BIT 6 IS THE ONE
0006 28FA       00340         JR      Z,VWAIT         ;TRANSMITTER BUSY
0008 79         00350         LD      A,C             ;GET CHARACTER
0009 D3EB       00360         OUT     (UDATA),A       ;SEND IT
000B E6FF       00370 PF      AND     255             ;LF/CR SWITCH
000D FE0D       00380         CP      0DH             ;CAR. RET?
000F 2004       00390         JR      NZ,VID010
0011 0E0A       00400         LD      C,0AH           ;INSERT L.F.
0013 18ED       00410         JR      VWAIT           ;SEND L.F.
0015 C1         00420 VID010  POP     BC              ;RECOVER CHARACTER
0016 F1         00430         POP     AF              ;RECOVER FLAGS
0017 C30000     00440 VCONT   JP      0000            ;PATCH POINT
                00450 ;
                00460 ;       NEW KEYBOARD DRIVER PATCH
                00470 ;
001A C5         00480 KEYBD   PUSH    BC              ;SAVE FLAGS
001B F5         00490         PUSH    AF
001C DBEA       00500         IN      A,(USTAT)       ;CHECK UART
001E E680       00510         AND     80H             ;CHECK BIT 7 = DATA REC'D
0020 280B       00520         JR      Z,KOUT          ;NONE READY
0022 DBEB       00530         IN      A,(UDATA)       ;GET DATA
0024 FE01       00540         CP      1               ;CHECK FOR BREAK
0026 2805       00550         JR      Z,KOUT          ;AND IGNORE
0028 4F         00560         LD      C,A
0029 F1         00570         POP     AF              ;RECOVER FLAGS
002A 79         00580         LD      A,C             ;CHAR. TO A
002B C1         00590         POP     BC
002C C9         00600         RET
002D F1         00610 KOUT    POP     AF
002E C1         00620         POP     BC
002F C30000     00630 KCONT   JP      0000            ;PATCH POINT
                00640 ;
                00650 ;       INSTALLATION OF NEW VIDEO AND KEYBOARD
                00660 ;       LINKS TO RS-232
                00670 ;
                00680 ;       FIRST, PATCH VIDEO DCB TO NEW ROUTINE
```

Listing 1 continued

INSTAL. The DI instruction contained there disables the interrupts to prevent the CPU from scanning the keyboard while you're changing things.

Since you want both your linking routine and the computer's ROM routine to either end with a return instruction or jump to the location the device control block intended before you started meddling, your routine must save the original address for each device control block. Line 710 picks up the original address from the video device control block, then line 730 stores the address of your video handler in the device control block. Line 740 loads the original address into VCONT+1 and finishes the video patch.

Lines 810–840 do the same for the keyboard. The remainder of the code, lines 890–930, configures the UART. Line 890 executes a master reset, lines 900–910 write a 55 hex (300 baud) to the baud register, and lines 920 and 930 set 7-bit words, even parity, 1 stop bit by writing an A4 hex to port EA hex. Line 940 starts the clock again and jumps the routine back to DOS.

The video and keyboard subroutines are at labels VIDEO (line 300) and KEYBD (line 480). Let's examine the video first, since it's the simpler of the two. When the CPU sends a character to the screen, it gets routed through the video device control block, which sends it to your linking routine. The character is in the C register when the routine gets to the video subroutine, and lines 300–310 save the status of the flags and the character.

The routine waits at V WAIT (line 320) for the UART to get ready to receive another character. IN A,(USTAT) gets the status byte from the UART status register, EA hex. Bit 6 is the Transmitter Empty flag; your routine is looking for a 1 in this position (see Table 1). The video subroutine masks out everything but bit 6 by ANDing the status byte with 40 hex (01000000 binary). If the AND command results in a zero, the routine loops back to V WAIT.

When the UART signals that it's ready to accept a byte, your routine loads the character from the C register to the A register and writes it to port EB hex. The UART sends the character out through port EB hex.

Some of the BBS's callers will be using a terminal program that requires a carriage return and a line feed, and some won't. You must offer line-feed after carriage-return as an option for the BBS.

The Basic program we've written to accompany this month's machine-language programs has a way to turn this option on or off. Basic can POKE a zero into PF+1 to turn off line feed

Listing 1 continued

```
                00690 ;
0032 F3         00700 INSTAL  DI                        ;A LITTLE PRIVACY, PLEASE
0033 ED5B1E40   00710         LD      DE,(VBLK+1)       ;GET EXISTING ADDRESS
0037 210000     00720         LD      HL,VIDEO          ;NEW ADDRESS
003A 221E40     00730         LD      (VBLK+1),HL       ;PUT NEW ADDRESS IN DCB
003D ED531800   00740         LD      (VCONT+1),DE      ;PUT OLD ADDRESS IN VIDEO
0041 00         00750         NOP                       ;TRSDOS ONLY
0042 00         00760         NOP                       ;DELETE FOR LDOS
                00770 ;
                00780 ;       THEN, PATCH KEYBOARD DCB
                00790 ;       TRSDOS 1.3 ONLY
                00800 ;
0043 ED5B1640   00810         LD      DE,(KBLK+1)       ;GET OLD ADDRESS
0047 211A00     00820         LD      HL,KEYBD          ;NEW ADDRESS
004A 221640     00830         LD      (KBLK+1),HL       ;INSTALL NEW ADDRESS
004D ED533000   00840         LD      (KCONT+1),DE      ;OLD ONE TO OUR ROUTINE
                00850 ;
                00860 ;       FINALLY, CONFIGURE UART FOR 300/7/E/1
                00870 ;       TRSDOS 1.3 ONLY
                00880 ;
0051 D3E8       00890         OUT     (0E8H),A          ;MASTER RESET
0053 3E55       00900         LD      A,55H             ;SET 300 BAUD SEND/RECV
0055 D3E9       00910         OUT     (0E9H),A
0057 3EA4       00920         LD      A,0A4H            ;7/E/1
0059 D3EA       00930         OUT     (0EAH),A          ;SET IT
005B FB         00940 IN010   EI
005C C32D40     00950         JP      TODOS
0032            00960         END     INSTAL
00000 Total Errors
```

End

Program Listing 2. Linking video, keyboard, and RS-232 in LDOS 5.1.3.

```
                00100 ;       TITLE   <LISTING 2 - LDOS>
                00110 ;       VIDEO, KEYBOARD AND RS232 LINKING
                00120 ;       LDOS 5.1.3
                00130 ;
                00140 ;
                00150 ;
401D            00160 VBLK    EQU     401DH             ;VIDEO DCB
4015            00170 KBLK    EQU     4015H             ;KB DCB
00EB            00180 UDATA   EQU     0EBH              ;UART DATA PORT
00EA            00190 USTAT   EQU     0EAH              ;UART STATUS REGISTER
00E8            00200 MSTAT   EQU     0E8H              ;MODEM STATUS PORT
0033            00210 PRINT   EQU     33H               ;ROM PRINT ROUTINE
002B            00220 KEYBRD  EQU     2BH               ;ROM KEYBOARD SCAN
0A9A            00230 BASIC   EQU     0A9AH             ;PASS TO BASIC
402D            00240 TODOS   EQU     402DH             ;JUMP TO DOS
                00250 ;
                00260 ;
                00270 ;
                00280 ;       NEW VIDEO DRIVER PATCH
                00290 ;
0000 F5         00300 VIDEO   PUSH    AF                ;SAVE FLAGS
0001 C5         00310         PUSH    BC                ;SAVE CHARACTER
0002 DBEA       00320 VWAIT   IN      A,(USTAT)         ;CHECK UART FOR CLEAR
0004 E640       00330         AND     40H               ;BIT 6 IS THE ONE
0006 28FA       00340         JR      Z,VWAIT           ;TRANSMITTER BUSY
0008 79         00350         LD      A,C               ;GET CHARACTER
0009 D3EB       00360         OUT     (UDATA),A         ;SEND IT
000B E6FF       00370 PF      AND     255               ;LF/CR SWITCH
000D FE0D       00380         CP      0DH               ;CAR. RET?
000F 2004       00390         JR      NZ,VID010
0011 0E0A       00400         LD      C,0AH             ;INSERT L.F.
0013 18ED       00410         JR      VWAIT             ;SEND L.F.
0015 C1         00420 VID010  POP     BC                ;RECOVER CHARACTER
0016 F1         00430         POP     AF                ;RECOVER FLAGS
0017 C30000     00440 VCONT   JP      0000              ;PATCH POINT
                00450 ;
                00640 ;
                00650 ;       INSTALLATION OF NEW VIDEO AND KEYBOARD
                00660 ;       LINKS TO RS-232
                00670 ;
                00680 ;       FIRST, PATCH VIDEO DCB TO NEW ROUTINE
                00690 ;
001A F3         00700 INSTAL  DI                        ;A LITTLE PRIVACY, PLEASE
001B ED5B1E40   00710         LD      DE,(VBLK+1)       ;GET EXISTING ADDRESS
001F 210000     00720         LD      HL,VIDEO          ;NEW ADDRESS
0022 221E40     00730         LD      (VBLK+1),HL       ;PUT NEW ADDRESS IN DCB
0025 ED531800   00740         LD      (VCONT+1),DE      ;PUT OLD ADDRESS IN VIDEO
0029 FB         00940 IN010   EI
002A C32D40     00950         JP      TODOS
001A            00960         END     INSTAL
00000 Total Errors
```

End

after carriage return, or a 255 to turn it on, depending on what the caller needs.

Consider how that works. ANDing the contents of the A register with 255 (11111111 in binary) doesn't affect it. If the character in the A register were a carriage return (0D hex) before the AND, it still will be. Then line 390 picks up the character, changes it to a line feed (0A hex), and jumps back to V Wait to send a line feed.

If Basic jams a zero into PF+1, the routine ANDs with zero (00000000 binary) to produce a result of zero. When line 390 detects a zero, it doesn't jump back to V Wait. Instead, lines 420 and 430 pick the character and the flags back from the stack, and line 440 jumps back to where the device control block intended for printing to the screen.

The keyboard patch is more complicated. When the computer arrives at your routine, the CPU hasn't scanned the keyboard yet. If the CPU gets a character from the UART and jumps to the routine originally contained in the device control block, the CPU scans the keyboard and loses the character received from the UART. Your keyboard patch needs two exits: 1 for the UART when it's holding characters and another for when it isn't.

According to Fig. 1, bit 7 of the UART status byte indicates whether or not a character is ready for the UART to pick up from port EB hex. Lines 480 and 490 save the flags and the BC register, then line 500 reads in the status byte and checks bit 7 by ANDing with 80 hex (10000000 binary).

If bit 7 is a zero, the routine jumps to KOUT and POPs AF and BC off the stack and jumps to the keyboard scanner. If bit 7 is a one, the UART contains a character that the routine picks up from port EB hex and puts in the C register.

Line 570 POPs the flags and line 580 moves the recovered character into the A register. After setting the stack with a POP BC, the routine returns to the operating system—just as if the CPU had scanned the keyboard.

If TRSDOS users press the break key and send a CHR$(1) character, they'll hang up the program. To account for this, line 540 of Listing 1 (the TRSDOS version) checks for a 1 and ignores it. LDOS avoids this problem by screening out the break key.

LDOS and TRSDOS are dissimilar operating systems. As you saw last month, LDOS lets you link devices but TRSDOS doesn't. Under LDOS, you can simply link the keyboard and the video to the communications line by typing:

```
LINK *KI *CL
LINK *DO *CL
```

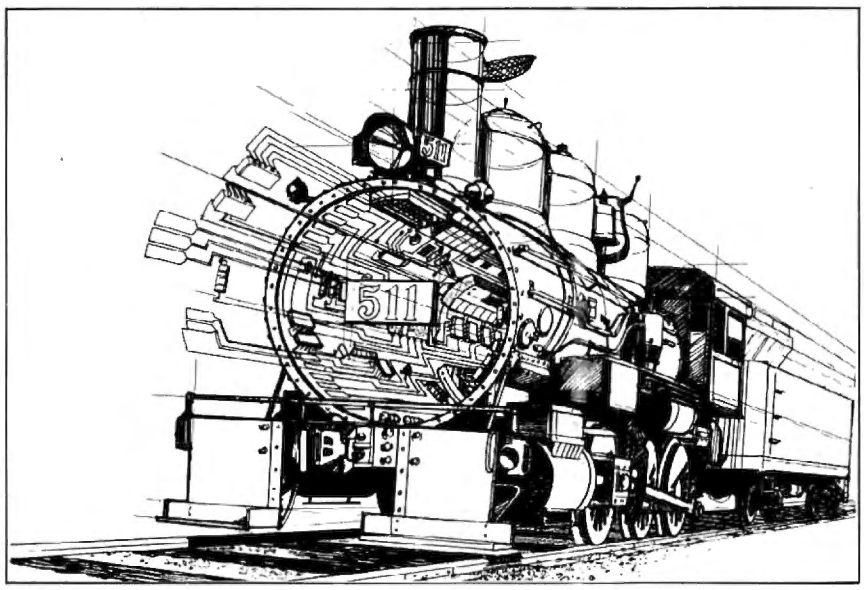
But this link won't accept a line feed after carriage return. You should link the video device to the communications line just as you did with TRSDOS, skipping the keyboard link. And using LDOS's Link command lets you take advantage of its type-ahead function.

The tracks are laid. Next month, you'll start building the BBS Express's engine. ■

Contact J. Stewart Schneider and Charles E. Bowen at P.O. Box 404, Catlettsburg, KY 41129.

Receiving Bytes With Your BBS

Nothing is as critical to the smooth operation of a computer bulletin board as its receiver. Writing the receiver in Basic is out, since a Basic program can't keep up with the speed of data exchange on a BBS. A receiver is the largest machine-code module in the program. We'll consider some of the problems it must handle.



Transmitting and Receiving

Compared to the task of receiving data, transmitting it from the TRS-80 is a piece of cake. All a transmitter must do is pump out bytes. The receiver must catch them and save them to disk. That takes time.

Periodically, the receiver sends a signal to the transmitter, telling it to stop transmission while it accesses a disk or does some other processing. Exchanging signals like this produces an effect called pipelining. (Pipelining also describes routing in a Xenix system, something unrelated to this article.)

Transmissions are electrical signals, and electricity moves so fast that it seems instantaneous. It's not. Imagine the transmitter as a baseball pitching machine, and the bytes as baseballs. The receiving machine is catching the baseballs, and periodically its receiving bin fills up. It must then signal the transmitter to stop. For this purpose the receiver is equipped with a red baseball.

When the receiver throws the red baseball to the transmitter, the transmitter must immediately stop throwing until the receiver throws a green baseball, or until a predetermined amount of time passes. If the time passes without a green baseball, the transmitter assumes that something is wrong, and terminates the transaction.

If the receiver waits until its bin is full before throwing the red ball, some of the baseballs are lost, because it takes time for the red ball to get to the transmitter and for the transmitter to react.

The transmitter continues to throw baseballs during this time, and baseballs in the air when the receiver throws the red ball are lost because the bin is full.

The receiver must therefore throw the red ball before its bin is full so that the bin can hold the balls in the pipeline.

In a telecommunications setup, the red ball is the XOFF signal, or DC3 (device control 3). You can send it from the keyboard by typing control-S. Its ASCII code is 19.

The green ball is a DC1, or XON. Send it by typing control-Q, or an ASCII 17. CompuServe and The Source subscribers will recognize these as the signals that freeze and unfreeze sceen displays.

The BBS's receiver section must constantly monitor its buffer and send an XOFF signal (red ball) before the buffer fills. Immediately on receiving the XOFF, the transmitter stops sending data. When the receiver processes the received data, it sends an XON signal (green ball), and the transmitter picks up from where it left off.

The data in the BBS must be easily accessible to Basic, so Basic must have access to the receiver buffer. A buffer is a contiguous area of memory where the program stores data, and it's the same as a Basic string. Our receiver stores characters received directly in a Basic string.

To use that string, Basic has to find the characters. A Basic program does so with the VARPTR command. VARPTR returns the address of a desired variable. In the case of a string, VARPTR returns the addresses of 3 bytes that define the string.

Basic can then pass this address to a USR call so the receiver can insert the received characters directly into the Basic string. The following is an example of how this works.

To see what VARPTR returns in the case of a string, see Program Listing 1. Line 10 defines Z$ as a string literal. Line 20 sets V equal to VAR-

The Key Box

Models I, III, and 4
48K RAM
Disk Drive

Program Listing 1. Demonstration program using the VARPTR command.

```
1 REM            LISTING 1
2 REM         STRINGS AND BUFFERS
10 CLS: DEFINT A-Y: Z$="RECEIVER BUFFER"
20 V = VARPTR(Z$): B = PEEK(V)
30 PRINT"STRING LENGTH IS";B
40 HL = PEEK(V+1) + PEEK(V+2)*256
50 FOR I = 0 TO B-1: PRINT CHR$(PEEK(HL+I));:NEXT
60 Z1$="BUFFER FILLED  "
70 FOR I =  0 TO B-1: POKE HL+I,  ASC(MID$(Z1$,I+1,1)):NEXT:
PRINT
80 LIST
```

End

Program Listing 2. The BBS Express receiver module.

```
                00100 ;         TITLE   <TOWNE CRIER MODULE>
                00110 ;         RECEIVER SECTION 3/23/84
                00120 ;         RECEIVES CHARACTERS FROM THE RS232 LINK
                00130 ;         OR THE KEYBOARD, AND STORES THEM
                00140 ;         SEQUENTIALLY IN I$ UNTIL THE LENGTH OF I$
                00150 ;         IS MET, OR A CARRAIGE RETURN IS ENTERED.
                00160 ;         RECOGNIZES BACKSPACE, AND RETURNS ERROR CONDITION
                00170 ;         ON LOSS OF USER CARRIER.
                00180 ;
                00190 ;
401D            00200 VBLK      EQU     401DH           ;VIDEO DCB
4015            00210 KBLK      EQU     4015H           ;KB DCB
00EB            00220 UDATA     EQU     0EBH            ;UART DATA PORT
00EA            00230 USTAT     EQU     0EAH            ;UART STATUS REGISTER
00E8            00240 MSTAT     EQU     0E8H            ;MODEM STATUS PORT
0033            00250 PRINT     EQU     33H             ;ROM PRINT ROUTINE
002B            00260 KEYBRD    EQU     2BH             ;ROM KEYBOARD SCAN
0A9A            00270 BASIC     EQU     0A9AH           ;PASS TO BASIC
402D            00280 TODOS     EQU     402DH           ;JUMP TO DOS
0001            00290 TRSDOS    EQU     1               ;SET TO 0 FOR LDOS
                00300 ;
                00310 ;
FE00            00320           ORG     0FE00H
FE00 CD61FE     00330 RECV      CALL    PARAM           ;GET STRING VALUES
FE03 06F5       00340 MN        LD      B,245
FE05 0E00       00350           LD      C,0             ;ACCUMULATOR
FE07 3E11       00360           LD      A,11H           ;CONTROL-Q
FE09 CD3300     00370           CALL    PRINT           ;SEND IT
FE0C DBE8       00380 MAIN      IN      A,(MSTAT)       ;CHECK MODEM STATUS
FE0E E620       00390           AND     32              ;CHECK CARRIER DETECT
FE10 2049       00400 TN        JR      NZ,NOTONE       ;LOST THE TONE
FE12 CD2B00     00410           CALL    KEYBRD          ;CHECK EVERYTHING ELSE
FE15 B7         00420           OR      A
FE16 28F4       00430           JR      Z,MAIN          ;NOTHING
FE18 FE08       00440           CP      8               ;BKSPCE?
FE1A 2010       00450           JR      NZ,NOTBKS       ;NO
FE1C 79         00460           LD      A,C
FE1D B7         00470           OR      A               ;AT FIRST CHAR?
FE1E 28EC       00480           JR      Z,MAIN          ;YES - IGNORE
FE20 04         00490           INC     B
FE21 04         00500           INC     B               ;FOR THE DECREMENT
FE22 0D         00510           DEC     C
FE23 0D         00520           DEC     C               ;FOR THE INCREMENT
FE24 2B         00530           DEC     HL
FE25 3620       00540           LD      (HL),20H
FE27 2B         00550           DEC     HL
FE28 3E08       00560           LD      A,8
FE2A 1801       00570           JR      NP              ;PERFORM BKSPCE
FE2C 77         00580 NOTBKS    LD      (HL),A          ;STORE CHARACTER
FE2D CD3300     00590 NP        CALL    PRINT           ;PRINT IT
FE30 23         00600           INC     HL              ;FOR NEXT CHARACTER
FE31 0C         00610           INC     C
FE32 FE0D       00620           CP      0DH             ;TERMINATOR ENTERED?
FE34 2802       00630           JR      Z,EXIT          ;TERMINATOR FOUND
FE36 10D4       00640 NOT010    DJNZ    MAIN            ;LOOP TIL DONE
FE38 3E13       00650 EXIT      LD      A,13H           ;CONTROL-S
FE3A CD3300     00660           CALL    PRINT           ;SEND IT
FE3D 06FF       00670           LD      B,255           ;DELAY
FE3F 1E0A       00680           LD      E,10            ;MAX # OF CHAR.
FE41 DBE8       00690 EXI010    IN      A,(MSTAT)       ;CHECK MODEM STATUS
FE43 E620       00700           AND     32              ;CARRIER DETECT
FE45 2014       00710 TT        JR      NZ,NOTONE       ;LOST CARRIER
FE47 CD2B00     00720           CALL    KEYBRD          ;CHECK FOR INCOMING
FE4A B7         00730           OR      A
FE4B 2806       00740           JR      Z,EXI030        ;NONE - EXIT
FE4D 0C         00750           INC     C               ;COUNT CHARACTER
FE4E 77         00760           LD      (HL),A          ;STORE IT
FE4F 23         00770           INC     HL              ;BUMP POINTER
FE50 1D         00780           DEC     E               ;DECREMENT CHAR. COUNT
FE51 2802       00800           JR      Z,EXI020        ;MAX # RECEIVED
FE53 10EC       00810 EXI030    DJNZ    EXI010          ;DECREMENT TIMER
FE55 69         00820 EXI020    LD      L,C
FE56 2600       00830           LD      H,0             ;TO PASS TO BASIC
FE58 C39A0A     00840           JP      BASIC
FE5B 21FFFF     00850 NOTONE    LD      HL,-1
FE5E C39A0A     00860           JP      BASIC           ;PASS ERROR TO BASIC
```

Listing 2 continued

PTR(Z$). PEEKing in location V provides the length of the string, which the program set to B. Line 30 prints it. That tells how long the string is, but we still don't know what the string contains.

PEEKing around the VARPTR of a string won't divulge what's in it, but it does indicate the string location in the next 2 bytes. The following 2 bytes contain the actual address of the start of the string in Z80 least-significant byte/most-significant byte (LSB/MSB) format.

Line 40 sets HL equal to the address of the start of the string. It PEEKs HL for the first character in the string. Line 50 runs a For...Next loop from zero to N − 1 and prints the ASCII representation of the contents of successive memory locations, starting at HL, demonstrating that it has found the string.

Run that same For...Next loop using POKE instead of PEEK, and the program changes the string. Line 70 POKEs the ASCII code of each letter from Z1$ into the same memory addresses PEEKed in line 50.

List the program, and you can see that we've actually changed the definition of Z$ in line 10 from receiver buffer to buffer filled. That's the same idea we use for the receiver, only in machine code.

Next month, when we get to the Basic code, we'll demonstrate how to pass a value from Basic to a machine-code subroutine. For now, we look at what happens from the Z80 perspective (see Program Listing 2).

Basic passes the VARPTR of the buffer string (I$ in the Basic program, which Basic defines as 255 blank spaces, the maximum length of a Basic string) to the machine-code receiver, RECV.

In line 330, RECV calls subroutine PARAM. At line 1660, PARAM calls 0A7F hexadecimal (hex), a documented ROM call that picks up the value passed by Basic (in this case, VARPTR(I$)), converts it to an integer, if necessary, and places it in the HL register pair. When the program returns from 0A7F hex, HL contains VARPTR(I$), the address of the definition bytes in the buffer string.

PUSH HL, POP IX transfers VARPTR(I$) to the IX register for doing indexed loading. The IX register is one of two indexed (or pointer) reg-

Listing 2 continued

```
FE61 CD7F0A    01660 PARAM   CALL    0A7FH          ;GET VARPTR(STRING)
FE64 E5        01670         PUSH    HL
FE65 DDE1      01680         POP     IX             ;TO IX
FE67 DD4600    01690         LD      B,(IX+0)       ;LENGTH TO B
FE6A DD6E01    01700         LD      L,(IX+1)
FE6D DD6601    01710         LD      H,(IX+1)       ;ADDRESS TO HL
FE70 C9        01720         RET                    ;DONE
0000           01730         END
00000 Total Errors
```

End

Sooner or later, somebody will call your BBS and the connection will break before the caller signs off.

isters on the Z80 chip. Loading an address into the IX register is like placing a book mark. You can manipulate other memory addresses from the book mark by adding or subtracting a number, called an offset, from the position of the book mark. It's easier to understand by looking at lines 1690–1710.

Assume that VARPTR(I$) equals E000 hex, so that IX contains E000 hex in line 1680. Line 1690 loads the B register with the contents of memory address E000 hex+0, or E000 hex. This is the same as PEEKing in VARPTR(I$) and loading the B register with the length of I$, as we did in line 20 of Listing 1. Remember that the LD instruction doesn't load B with E000 hex, but with the contents of the address pointed to by the IX register plus the offset, zero in this case.

You also need the start address of the string. As in line 40 with the Basic variable HL, load the HL register pair with this address, again using indexed addressing. Line 1700 loads the L register with the contents of E000 hex+1, or E001 hex, the LSB of the string's starting address.

After the LD L,(IX+1) command, the IX register still contains E000 hex, but the L register contains the contents of memory address E001 hex. LD H,(IX+2) loads the MSB of the string's starting address into the H register from the contents of memory address E000 hex+2, or E002 hex. HL now has the address of the first character in I$, and B has a length of I$. PARAM is finished, so the program returns.

After it calls PARAM, line 340 puts a 245 in the B register, destroying the value picked up from VARPTR(I$). We do this because other routines call PARAM to get the length of the string, which can be different from 255. At RECV, the string is always 255 characters, so the value returned in B isn't needed.

If the string is 255 characters long, why set B to 245? In a word, pipelining. We want to send an XOFF while there's still space in the buffer to catch the characters in the pipeline.

We have allowed for a 10-character overflow. When the program receives 245 characters, it sends an XOFF, then monitors transmissions until they stop, saving the overflow characters in the last 10 spaces of the buffer.

When the program returns to Basic, it needs to know how many characters it received. We use the C register as an accumulator to count the characters. Line 350 sets the C register to zero. Line 360 loads an XON into the A register by setting it to 11 hex, or 17 decimal (the green baseball in our earlier example). A Print command sends an XON signal to the transmitter through the link between the video and the communications line. This is the start signal.

RECV has one other task. Sooner or later, somebody will call your BBS and the connection will break before the caller signs off. If this happens, the Towne Crier must detect it and bring the board to a normal close for the next caller. That's the purpose of lines 380–400, and the NOTONE section.

The modem status port, port E8 hex, returns the state of the carrier in bit 5. If the caller's modem is sending a carrier tone, bit 5 is a zero. If not, it's a 1. Line 390 masks out all but bit 5 by doing an AND with 32 (00100000 in binary), and line 400 checks for a non-zero condition. If the AND results in a non-zero number, control passes to NOTONE, which loads the HL register with a −1 and passes this back to Basic. Basic tests for a −1 from NOTONE and resets the board.

In some installations, the carrier detect line from the modem does not connect to the computer. That saves wire, but plays havoc with the remote installation. The BBS software must have access to the carrier detect line from your modem.

Sometimes, you won't want to test for NOTONE. The Sysop will often run the board without a modem connected for file maintenance. Basic can POKE zeros, the Z80 NOP instruction, into locations TN, TN+1, TT, and TT+1 to eliminate the tests for no carrier, and the Sysop can go about his business with the modem off.

RECV's main job is to receive from the keyboard or the communications line. Last month, we linked the keyboard scanning routine to the communications line in such a way that every time the program scanned the keyboard, it scanned the communications line first. In this way, the program can check for data over the communications line or from the keyboard by a single call to the keyboard scanner, KEYBRD, in line 410.

Any character received from the communications line or the keyboard returns in the A register. If no character is available in either place, the A register contains a zero. Line 420 Ors A with itself. If A is zero, the Or sets the zero flag, and line 430 sends control back to Main to try again.

If A does not contain a zero, the Or won't disturb it, but processing falls through to line 440, which checks for the backspace key. If the character is a backspace, and if it is at the first position of the string, the program does nothing (it can't back out of the buffer). Lines 460–480 check for this, and jump back to Main if the program is at the first character in the buffer.

The remaining code in lines 490–570 backs everything up one character, reloads the backspace character into A, and jumps to NP to print it.

If the character isn't a backspace, the program jumps to NOTBKS. The HL register still contains the address of the first character in I$, so the LD (HL),A instruction loads the character

We've got two things to worry about: receiving too many characters, and timing.

received from the A register to the address contained in the HL register.

The first character is in I$, so the program invokes a Print command to put it on the screen. The screen links to the communications line, so the Call to Print also echoes the character over the communications line for full-duplex transmission.

The program stores the next character in the next consecutive memory location, and increments HL by an INC HL to point to the right address, bumps C by one to count the character, and checks for a carriage return.

A carriage return is the termination character. If the character is not a carriage return, the DJNZ in line 640 decrements the B register, tests for non-zero, and jumps back to Main if the program hasn't received 245 characters.

If Towne Crier counts B down to zero, or if it receives a carriage return, control goes to Exit. This is where the program takes care of pipelining. First, we send an XOFF by printing a DC3 or by typing a control-S.

We've got two things to worry about here—receiving too many characters, and timing. The program exits to Basic when it has received the remaining 10 characters, or when time is up. The B register keeps the time out, and the E register keeps the character count. The program loads E with 10, and B with 255, a purely arbitrary value.

The loop from EXI010 to EXI030 works like the main loop, and counts the characters in the C register. It falls through either when it receives 10 characters, or when the B register counts down to zero.

The program transfers the contents of the C register to the L register at EXI020, sets H to zero, and jumps to Basic by a JP 0A9A hex. This jump passes the value of the HL register to Basic, so the Basic program knows how many characters it has received.

Once control returns to Basic, I$ contains the characters received. If the Basic call is something like CT = USR0(VARPTR(I$)), the program sets CT to the number of characters received, or to −1 if the connection is broken.

In next month's BBS Express, we start looking at the BBS's Basic code. ■

The BBS Express, 80 Micro*'s bulletin board system, is open 24 hours a day. Call us at 603-924-6985 to see the finished product.*

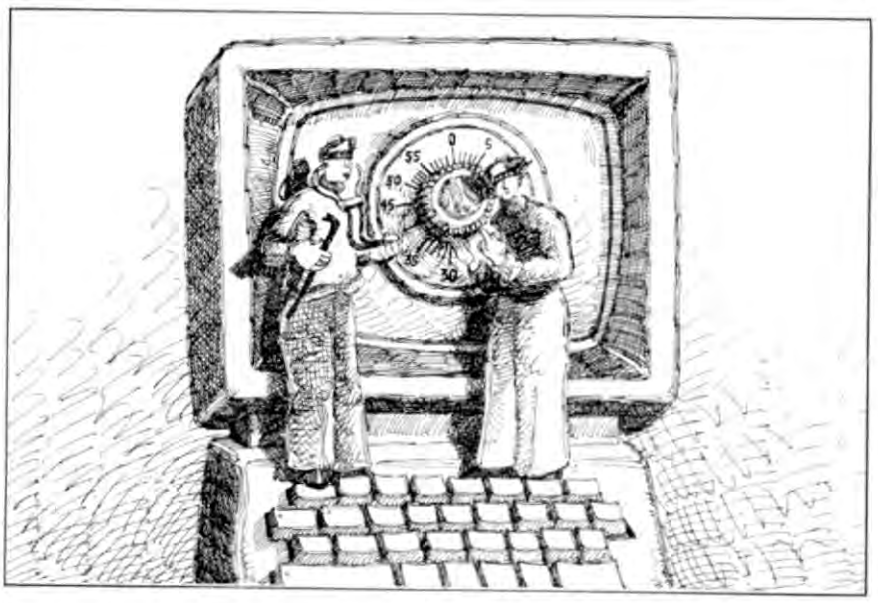
You can reach J. Stewart Schneider and Charles E. Bowen either through their bulletin board at 606-739-6088 or c/o Saturday Software, P.O. Box 404, Catlettsburg, KY 41129.

See List of Advertisers on Page 194

Controlling the Flow: The BBS File Structure

In previous columns we've discussed communications theory and developed the Assembly-language communications module for the BBS Express. Eventually, you'll write the program that lets Basic interact with that Assembly code.

This month, though, we're going to set aside communications theory and begin building the bulletin board structure. To do this, you might need some background on file management in Basic.



Basic Data Files

In the BBS Express, a Basic program handles the greeting, file management, and the machine/user interface. Using the groundwork already laid, Basic hosts the bulletin board. We'll review how Basic treats data files.

You can store data in Basic as a sequential file or as a random-access file. You use both file types in your BBS, so if you're unsure about this aspect of Basic programming, this explanation should help you.

Sequential Files

A sequential file starts reading or writing from the beginning of a line and finishes reading or writing at the end of a line. To retrieve information from the middle of a sequential file, your program must read it from the beginning. To change the information, you read the whole file into memory, make the changes you want, and then write the file back to disk.

For our BBS, sequential files are too inconvenient for storing items that you want to change, such as a user log or list files. However, sequential files are excellent for storing information of uncertain length that doesn't change, like messages. That's just how we use them, as you'll see in the Program Listing.

Random Files

Random files consist of many records of equal size that you can select, bring into memory, and change without disturbing the rest of the file. They're handy for logs and lists, like headers on messages or data files, but they're impractical for storing text because their logical record length is always the same.

A text file, such as a message on the BBS Express, probably isn't an even multiple of the logical record length, so random files would waste disk space.

Another reason for not using random files for storing text is the Field statement. A single record in a random file can contain several pieces of information. The first 20 bytes in the record might contain a name, the next 25 an address, and so on.

Basic uses the Field statement to assign field names to the space in a record so that it can manipulate the information. For example, look at the BBS Express's Field statements in the Listing. Imagine how convoluted the code would become if you stored the text in a random file.

LDOS vs. TRSDOS

This brings us to a difference between the two operating systems we've chosen to work with, LDOS and TRSDOS. Using LDOS, you can open an existing sequential file for random access, pull something out of the middle of the file, change it, and then put it back.

TRSDOS, however, can't do this because of a problem with the logical record length. TRSDOS assumes that the logical record length of all files is 256 bytes long (255 on a Model I), unless you specify variable-length files by answering the Basic prompt, "How Many Files?" with a number and the

The Key Box LOAD 80

Model III
48K RAM
Two Disk Drives

letter V (for variable). If you fail to do so, Basic can't handle a logical record length of anything other than 256 bytes.

If you do specify variable-length records, TRSDOS 1.3 lets you specify the logical record length for the file when you create it. Thereafter, you have to specify the same logical record length to get at the file. If you specify a logical record length different from that used at the file's creation, TRSDOS will give you strange results, ranging from refusing to retrieve anything from the file to generating an error.

LDOS is much more flexible. It assumes that you want to use variable-length records unless you go to the trouble to tell it otherwise. LDOS lets you open a file with a logical record length different from that set at creation, and still handles everything correctly.

Headers

Computer bulletin boards like the BBS Express are, among other things, message-switching programs. The BBS you're writing here stores and displays messages from one user to another. Each message on the message board needs a header to identify it. Each message also needs a place to store the text of the message.

Our headers contain four pieces of information for you, and an additional two pieces of invisible, secret information used by the system. You can change the invisible information, so you should make the file for headers, Messages/BBS, a random file. The key to that decision is that the information is changeable.

Because the text of the messages remains unchanged and is of uncertain length, we assigned the information to a sequential file. The BBS represents each message on the board by a sequential file containing the text, and by a record in a header file, Messages/BBS.

The Table sets out the field names in the records of all of the BBS's files. The Listing shows the Open and Field statements for all files.

Messages/BBS's line 220 opens the file, and sets the logical record length to 97. Why 97? Because the information we keep in each record for each message on the board equals 97, so Messages/BBS's logical record length is 97.

The reference to SD$ probably needs some explanation. The BBS is designed to work with a two-drive system, and you need to specify the drive where a given file can be found. Earlier in the program (in line 100 for those with a copy), SD$, the system drive, is set to drive zero and DD$, the data drive, to drive 1. Line 220 Opens Messages/BBS, on drive zero. If you want to use another drive, change the definition in line 100.

The first four fields in a record from Messages/BBS are self-explanatory. But what about the secret information, F2$ and S2$?

Invisible Files

F2$ is the file name of the sequential file that contains the text of the message. F2$ safely tucks this file's name away, never displaying it to the user.

The secret files protect you from telecommunications vandals, people who delight in leaving messages named "System" in the hope of zapping a board.

However, only the BBS Express can put a file name on the disk's directory.

```
190 OPEN"R",3,"SYSTEM/BBS"+SD$
200 FIELD 3,2 AS SL$,2 AS SH$,2 AS SN$,2 AS SV$,16 AS SP$,1 AS SF$
,16 AS SA$,2 AS SC$,2 AS SM$,2 AS NM$,2 AS ND$,2 AS DS$:RETURN
210 FOR X=0TO15:FIELD 3,X*16 AS DUMMY$,16 AS F$(X+1):NEXT:RETURN
220 OPEN"R",1,"MESSAGES/BBS"+SD$,97
230 FIELD 1,20 AS T1$,20 AS F1$,32 AS S1$,17 AS T2$,7 AS F2$,1 AS
S2$:RETURN
240 OPEN"R",2,"MEMBERS/BBS"+SD$,113
250 FIELD 2,20 AS N1$,25 AS A1$,25 AS C1$,1 AS L1$,2 AS V1$,16 AS
A2$,2 AS M1$,2 AS C2$,16 AS P1$,2 AS UH$,1 AS LP$,1 AS RP$:RETURN
260 OPEN"R",2,"XASPACE/BBS"+SD$,181
270 FIELD 2,20 AS N1$,16 AS P1$,128 AS D1$,9 AS F2$,2 AS XA$,4 AS
XL$,1 AS LP$,1 AS RP$:RETURN
280 OPEN"R",3,"USER/BBS"+SD$,62
290 FIELD 3,20 AS U1$,17 AS U2$,17 AS U3$,8 AS U4$:RETURN
```

Program Listing. Program for storing information in sequential files.

File: System/BBS Lines 190–210

Field Variable	Contents
SL$	Low message number on system
SH$	High message number on system
SN$	Number of message on system
SV$	Default video width
SP$	System password
SF$	System line feeds
SA$	Default access
SC$	Number of callers to system
SM$	Maximum number of messages on system
NM$	Number of members
ND$	Number of data files
DS$	Data slot
F$(	Section names
NM$	Index string

File: Messages/BBS Lines 220–230

Field Variable	Contents
T1$	To whom addressed
F1$	From whom
S1$	Message subject
T2$	Date and time sent
F2$	Invisible file name
S2$	Bit-mapped

File: XASPACE/BBS Lines 260–270

Field Variable	Contents
N1$	XA file's public name
P1$	XA file's password
D1$	Description of file
F2$	XA file's secret name
XA$	Number of times accessed
XL$	File's length in bytes
LP$	Left pointer
RP$	Right pointer

File: Members/BBS Lines 240–250

Field Variable	Contents
N1$	Member's name
A1$	Street address
C1$	City/State/Zip
L1$	Line feed after carriage return
V1$	Video width
A2$	User's authorized access
M1$	Number of messages left
C2$	Number of calls
P1$	User's password
UH$	High number retrieved
LP$	Left pointer
RP$	Right pointer

File: User/BBS Lines 280–290

Field Variable	Contents
U1$	User's name
U2$	Time in
U3$	Time out
U4$	Elapsed time

Table. Field variables.

The vandals can leave messages named whatever they wish, but these names never get on the disk's directory.

What file name should the BBS Express use? We assigned each message a number. We construct a unique file name using the message number, which stores the text. For example, the text of message 2 will be stored in a sequential file named MSG0002/BBS, message 100 in MSG0100/BBS, and so on.

The final field is S2$. This is bit-mapped, like the bit-mapped port we talked about at the start of this series (see "The BBS Express," May 1984, p. 42). Each bit in the byte is assigned a different meaning. Next month, we'll show you how to map information into S2$ and then retrieve it.

In addition to the message board, the BBS Express offers a data base of permanent files of interest to its users. This area (often referred to as the XA space, perhaps in deference to CompuServe Information Service, which uses the same term for data base) requires the same kind of management as the message board. Lines 260 and 270 field the records in the header file, XASPACE/BBS.

Vandals can leave messages named whatever they wish, but these names never get on the disk's directory.

There's some invisible information here as well. The real name of the sequential file containing the text is hidden in F2$, and there are two mysterious pointers, labeled Left and Right. The pointers work with a binary tree which you'll construct in a future issue. You can see the same fields in the membership log, lines 240–250.

That leaves two files, User/BBS and System/BBS. User/BBS stores the names and log-in and log-out times for your visitors. It's a random file. Periodically, you can kill this file when it gets too long.

System Defaults

The BBS Express's bulletin board lets you set certain system defaults, such as the line width, establishing a line feed after a carriage return, the names of the sections of the board (more on this in a later issue), and various other parameters that the BBS needs for a new user. This is stored in System/BBS, along with an in-memory string that Record 1 is fielded one way (line 200), Record 2 in another (line 210), and Record 3 still another (various program lines not listed here). You get a lot of information into just three System/BBS records.

Catch the BBS Express next month and we'll explain an advanced level of secret-keeping—bit mapping. ■

Contact J. Stewart Schneider and Charles E. Bowen at P.O. Box 404, Catlettsburg, KY 41129.

✓ See List of Advertisers on Page 162

Storing and Retrieving Messages With Your BBS

One advantage of a BBS is that you can satisfy a wide variety of user interests by setting aside different areas of the bulletin board as special-interest sections. For instance, you can establish one section for person-to-person messages, an area for owners of particular computers, a section for public-domain programs, and so on. These divisions allow the user to read only the messages in a desired section. This benefits both the user, who can skip sections of the board of little interest, and to the sysop, who can restrict access to some sections, reserving space for private use.

Last month we introduced the idea of these message centers and described how message information was stored in two places, one for the section number where a message was stored (S2$), the other for the actual text of the message. S2$ is bit-mapped, and each bit of the 8-bit byte in S2$ stands for something different.

We've divided the BBS Express's message board into 15 separate sections, represented in S2$ as 0123456789ABCDE. This way, you can tag messages for specific sections of the board.

Messages/BBS, our file of headers, stores the section number of an incoming message. Because you can store a number from zero to 15 using only 4 bits of binary data, we'll use the first 4 bits of each byte in S2$ (bits 0–3) to store the message's section number.

Fortunately, that leaves 4 bits available for other uses. Because you have to know whether or not an addressee has received his message, you can assign that task to bit 4 by using the following code: 1 = message received, 0 = message not yet received. Thus, if bit 4 is a zero, the addressee hasn't gotten his message.

Since privacy is sometimes necessary on a bulletin board, BBS Express lets you store private messages which only the sender, the receiver, and the sysop can access. You can ensure this

by setting up the code 1 = private, 0 = public and storing that code in bit 5.

Bits 6 and 7 are left unassigned and are available for future expansion. One possibility is to mark a message as permanent so that it would never be scrolled off the board. Another possibility is to determine whether or not there's a reply to a message.

Bit-Mapping and Binary Format

While it's true that bit-mapping in Basic requires a large code overhead, it is an important technique for any Basic programmer. We're using bit-mapping in BBS Express for tutorial value as well to provide a practical and efficient way to store message information. To better understand bit-mapping in Basic, first consider the system in binary format.

Start by setting S2$ to CHR$(0), which is CHR$(00000000) in binary. This allows the variable to make fresh assignments for each new message. The message is assigned a section number between 1 and 15, which in binary is a number between 00000001 and 00001111.

A logical OR can include the section number in S2$ without disturbing the other bits. You do this in Basic by executing a logical OR with the ASCII value of S2$ and the number of the specific section, N%. For example, S2$ = CHR$(ASC(S2$) OR N%).

How does this work? First, assume that N% is 10 (00001010 binary), and S2$ is 0 (00000000 binary). In binary, ORing N% and S2$ is 00000000 OR 00001010 or 10 decimal. Thus you've merged the section number into S2$.

Although you could have achieved the same result by simply executing S2$ = CHR$(N%), the technique is what's important as you'll use it later to merge other information into S2$.

We mentioned earlier that to make a message private, you set bit 5 of S2$ to 1. This is done with a logical OR as

The Key Box

Model III
48K RAM
Disk Basic
Two Disk Drives

well. This byte, with only bit 5 set (00010000), translates to 16 decimal. If you execute a logical OR with the ASCII value of S2$ and 16, all the bits in S2$ will be unaffected, except for bit 5, which is set.

In binary format, S2$ is now CHR$(00001010). 00001010 OR 00010000 = 00011010, or 26 decimal.

The Private flag is set, while the section number remains unchanged.

In a similar fashion, let bit 4 be responsible for marking a retrieved message. Set bit 4 by executing an OR with 32 (00100000 binary). S2$ = CHR$ (ASC(S2$) OR 32). In binary, 00011010 OR 00100000 = 00111010, or 58 decimal.

Other methods exist to do the same thing. For example, you can get the same result with S2$ = CHR$(ASC (S2$) + 10 + 16 + 32). But, because you'll be using logical ANDs to retrieve the information stored in S2$, you should use logical ORs to store it. In other words, what OR has wrought, AND will unwrought.

Start with S2$ = CHR$(58), 00111010 in binary. To recover the section number N%, use AND with the ASCII value of S2$ and 15 (00001111 binary). For example, N% = (ASC(S2$) AND 15), 58 AND 15 = 10, our section number.

To print the message, you first must recover it from the secret file.

Notice that bits 1–3 are unaffected, while bits 4–7 are bypassed. If you turn this around to 11110000, or 240 decimal, you can mask out bits 1–3 and isolate the top 4 bits for manipulation. Looking at this example in binary: 58 is 00111010. 00111010 AND 11110000 = 00110000, or 48 decimal.

Storing and Retrieving Information

All that remains now is to insert a code that transfers information to and from the disk where the BBS stores it. Program Listings 1 and 2 do just this: Listing 1 loads information onto the disk, while Listing 2 removes information from the disk.

In Listing 1, for example, line 3280 calls line 3290 as a subroutine. This lets the caller leave a message in one of two ways: with the L (Leave) command, or by replying to a displayed message. By creating a subroutine out of lines 3290–3640, you can call information from two places in the program.

Line 3280, the routine for leaving a message, begins by setting S7$ to CHR$(0). If line 3290 is called from a Reply command, S7$ will already contain the high 4 bits from S2$.

Throughout the program, GOSUB 130 acts as the call to the machine-code Receive section (RECV). GOSUB 130 returns with count equal to the number of characters entered over the communications link or keyboard, while I$ contains the characters.

Lines 3320–3340 prompt the section number of the message, and assign S6$ to the digit of that section. S9$ receives the number of the section, which corresponds with S6$.

Line 3610 LSETs the information into the file fields and places the assembled record into Messages/BBS, the message log, at record SN, the variable holding the number of messages on the system.

Program Listing 2 recovers the information. The header printing routine will then use this information, which we'll look at more extensively next month.

To print the message number, you first must recover it from the secret file constructed last month. This is done by setting N = VAL (RIGHT$(2$,4)). For example, if the file name is MSG0110, then N would be 110.

Since the "To," "From," and "Subject" sections of the message prompt are strings, no conversion is necessary. It is necessary, however, to assign variables so that you can manipulate them later. The variables T$, TT$, and S8$ are assigned the values To, From, and Subject, respectively.

Because of the mask with 240, S9$ gets the section number from S2$. You'll want the BBS program to print the digit of the section number, rather than its value. In our example, the section number was 10. Therefore, you want to print "Section: A" not "Section: CHR$(10)". Because CHR$(10) is a line feed, there's no need to print it.

Using Basic's MID$ function, S6$ retrieves the digit to print. You can see

Program Listing 1. BBS module for writing messages to disk.

```
3280 GOSUB220:S7$=CHR$(0):GOSUB3290:GOTO1720
3290 IF LEN(MN$)=254 THEN PRINT"Message board filled.  Please call
 back":RETURN
3300 PRINT"To: ";:GOSUB130:PRINTCHR$(17):IF CT=0 THEN RETURN ELSE
TT$=LEFT$(I$,CT)
3310 PRINT"Subject: ";:GOSUB130:PRINTCHR$(17):S8$=LEFT$(I$,CT)
3320 PRINT"Section:(Tap ENTER for list) ";:GOSUB130:PRINTCHR$(17)
3330 IF CT=0 THEN GOSUB830:IFCT=0THEN 3320
3340 S6$=LEFT$(I$,1):S9$=CHR$(INSTR("0123456789ABCDE",S6$)):IF (S9
$>CHR$(0))AND(INSTR(UA$,S6$)>0) THEN PRINTS$(ASC(S9$))ELSE 3320
3610 LSET F1$=NA$:LSET T2$=TIME$:LSET F2$=A$:LSET T1$=TT$:LSET S1$
=S8$:LSET S2$=CHR$((ASC(S7$) AND 32) OR (ASC(S9$))):PUT 1,SN:PRINT
"Awaiting delivery.":GOTO3640
3640 POKE MN,250:RETURN
```

End

Program Listing 2. BBS module that calls messages to the screen.

```
346 REM          DECODE MESSAGE HEADER
347 REM          RECORD IN BUFFER ON ENTRY
348 REM          EXIT: T$="TO:"
                       TT$="FROM:"
                       S8$="SUBJECT"
                       S9$=SECTION #
                       S6$=SECTION DIGIT
                       S7$=PUBLIC/PRIVATE RETRIEVED/NOT
350 E$=INKEY$:N=VAL(RIGHT$(F2$,4)):T$=T1$:TT$=F1$:S8$=S1$:S9$=CHR$
(ASC(S2$) AND 15):S7$=CHR$(ASC(S2$) AND 240):S6$=MID$("0123456789A
BCDE",ASC(S9$),1)
420 RETURN

1221 REM
1222 REM       Decide if printer is ready
1223 REM
1230 PR=(PR)AND((PEEK(&H37E8)AND240)=48):RETURN
```

End

this process in line 350 of Listing 2. In our example, S2$ is assigned the 10th character in the string 0123456789 ABCDE, which is A.

The high 4 bits of S2$ will go to S7$ for the header printing routine to use. This indicates the status of the message, i.e., whether the message is private or public and whether or not it's been retrieved by the addressee.

If ASC(S7$) AND 32 is 32, the message is private and the program will print a P. If the ASCII value of S7$ AND 16 is 16, then the addressee has retrieved the message, and the program will print an (X).

The printing routine uses a Boolean variable, PR, to indicate whether the output will go to the printer or the screen. If it goes to the screen, it is then transmitted through the communications line to the caller.

The sysop may elect to have output directed to the printer rather than to the screen with the sysop's Hard Copy option or by invoking the P command after a message appears. Choosing the Hard Copy option from the sysop's menu will set PR to true (−1). This works, assuming that the printer works properly. Line 1230 checks the printer status before allowing output to the printer.

Memory address &H37E8 is the printer status location. A PEEK to this address returns a byte that represents the current state of the printer. Because you're concerned here with the top 4 bytes, you execute an AND with 240. If the statement (PEEK (&H37E8) AND 240)=48) is true AND if PR is true (−1), the message is printed to the line printer instead of the screen.

If the printer isn't ready, ((PEEK (&H37E8) AND 240)=48) will not be true. In this case, PR will go to zero and the message will be printed to the screen.

In next month's BBS Express, we'll deal more with printing, so stay tuned. ■

The BBS Express, 80 Micro*'s bulletin board system, is open 24 hours a day. Call us at 603-924-6985 to see the finished product.*

You can reach J. Stewart Schneider and Charles E. Bowen either through their bulletin board at 606-739-6088 or c/o Saturday Software, P.O. Box 404, Catlettsburg, KY 41129.

✓ See List of Advertisers on Page 176

A Closer Look at Messages And Data-Base Files

Your message board is looking good; it can now accommodate a range of special-interest topics and handle specific requests. Last month's BBS Express gave you the code to read and write headers. This month we'll discuss messages and data-base files in detail.

First, we'll clear up a few problems concerning storage. Your BBS stores headers of fixed length in a random-access file. Because messages are also of fixed length (20 lines of 64 characters), you can store them in the same manner.

Data-base files, on the other hand, aren't of fixed length; their size is limited only by disk space. Therefore, you can't store them in random-access files. Instead of writing two routines, one for data-base files and another for messages, we've decided to simplify matters (and save disk space) by storing them in individual sequential ASCII files.

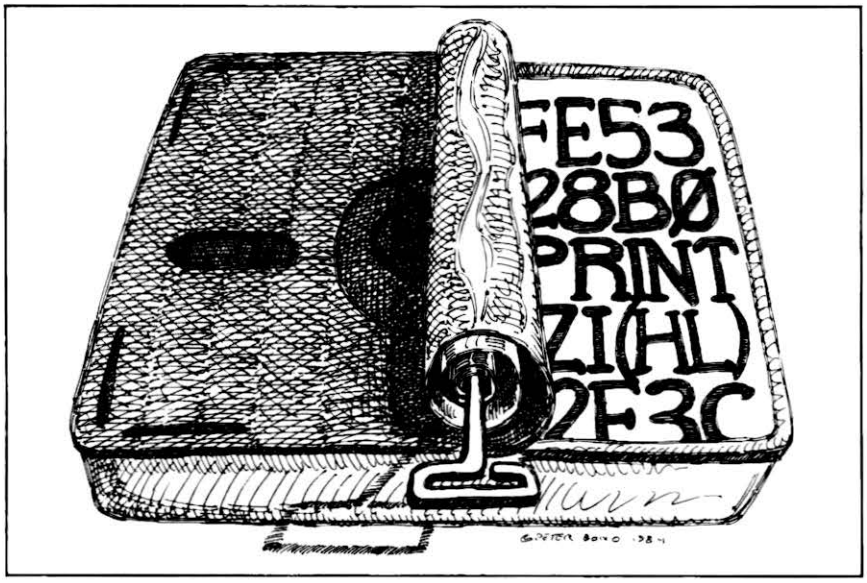


Illustration by Peter Bono

Carriage Returns

Sequential disk files become slightly more complicated where carriage returns are concerned. The PRINT# (file number) command writes data into a sequential file, while the INPUT# or LINE INPUT# commands retrieve data. PRINT# works with the disk the way the Print command works with the screen. That is, if there's a semicolon, the BBS program doesn't print a carriage return. If, on the other hand, a semicolon doesn't follow, the program adds a carriage return.

Look, for instance, at the string in the following example:

```
10 OPEN"O",1,"TEST/DAT:0"
20 LET A$="Hi there, sports fans"
30 PRINT#1,A$
40 CLOSE
50 END
```

This prints "Hi there, sports fans (CR)" on disk. If you insert a semicolon at the end of line 30, the program prints the string without the carriage return.

The following example removes the message from the disk and puts it onto the screen:

```
10 OPEN"I",1,"TEST/DAT:0"
20 INPUT#1,A$,:PRINTA$
30 CLOSE
```

This listing prints only "Hi there," because INPUT#1 stops inputting at the comma, which is a string terminator. If you change line 20 to LINE INPUT#1,A$, the program prints the entire string. This is important because the BBS messages are divided into paragraphs, each with a carriage return at the end.

So that Basic won't interpret an unwanted carriage return, we chose a substitute, CHR$(141), because it can't be sent from a remote system. The program sends messages sent to disk with PRINT#, changing all the carriage returns to CHR$(141)s.

One other problem surfaces when you use carriage returns as line terminators. This exists when a caller enters a carriage return for another line after reaching the 64-character limit. It's necessary to differentiate between a carriage return that signals the end of a line and one that signals the end of a paragraph.

The BBS Express does this by considering the indentation. For example, the program checks to see if the next line of text is indented. If it is, the program writes the carriage return in the previous line to disk as CHR$(141). If the line isn't indented, the program changes the carriage return to a space. The resulting string is written to disk with a PRINT# command, followed by a semicolon, and read in with a LINE INPUT# command. You can see this at work in line 3420 of Program Listing 1.

Writing a Message to Disk

Listing 1 is what remains of the writing routine started last month.

The Key Box LOAD 80

Model III
48K RAM
Two Disk Drives
Disk Basic

Program Listing 1. BBS module for writing data-base files and messages to disk.

```
3350 PRINT"Correct (y/n)?";:GOSUB130:PRINTCHR$(17)
3360 IFASC(I$)=78 THEN 3290
3370 IFASC(I$)<>89 THEN 3350
3380 CLS:PRINTCHR$(12);"Please enter your message now.":PRINT"Up t
o 20 lines, 64 characters per line.":PRINT"Enter a blank line to s
ignal the end of message":S=1
3390 TL=0:NC=-1:POKE M9,64:'   set max line length
3400 FOR LN=STO20
3410 PRINTCHR$(17):PRINTLN;": ";:GOSUB130:IF CT>64 THEN PRINTCHR$(
17);"No more than 64 characters per line, please":GOTO 3410
3415 IF ASC(I$)=141 THEN 3450
3420 IF LN>1 THENCR=INSTR(MG$(LN-1),CHR$(141)):IF ASC(I$)<>32 AND
CR>0 THEN MID$(MG$(LN-1),CR,1)=CHR$(32)
3430 LSET MG$(LN)=I$:MG(LN)=CT
3440 NEXT LN
3450 PRINTCHR$(17):LN=LN-1:PRINT:PRINT"Leave Options: "
3460 NC=0:TL=-1:PRINT"Subcommand (? for HELP): ";:GOSUB130:PRINTCH
R$(17):IF CT=0 THEN 3460
3470 C$=LEFT$(I$,CT)
3480 ONINSTR("SLRADCSP",C$) GOTO3520, 3650, 3660, 3620, 3660,3510,
3500
3490 PRINT"? - Prints this list":PRINT"S - Stores message":PRINT"S
P - Store Private Message":PRINT"A - Abort message":PRINT"R - Repl
ace Line":PRINT"L - List Message":PRINT"D - Delete line":PRINT"C -
 Continue entering text":GOTO3460
3500 S7$=CHR$(32):GOTO3520
3510 S=LN+1:IF S>20 THEN 3460 ELSE CLS:PRINTCHR$(12):GOTO3390
3520 IF B THEN RETURN
3530 M2=M2+1:SH=SH+1:IF SH>9999 THEN SH=1
3540 MN=SH:A$="MSG0000/BBS"+DD$:GOSUB860
3550 OPEN"O",3,A$
3560 FOR Z=1TOLN
3570 IF MG(Z)>0 THEN PRINT#3,LEFT$(MG$(Z),MG(Z));
3580 NEXT Z:PRINT"Message";SH;"stored...":CLOSE 3
3590 SN=SN+1
3600 MN$=MN$+MKI$(VAL(N$))
3620 PRINT"Abort (y/n)?";:GOSUB130:PRINTCHR$(17);
3630 IF ASC(I$)=78 OR (ASC(I$)<>89) THEN 3460
3650 FOR Z=1TOLN:PRINTZ;": ";LEFT$(MG$(Z),MG(Z)):NEXT Z:GOTO3460
3660 PRINT"Enter line #":GOSUB130:IFCT=0THEN3460
3670 Z=VAL(I$):PRINTCHR$(17):IF Z<1 OR Z>LN THEN 3460
3680 CLS:PRINTCHR$(12);"Line currently reads: ":PRINT:PRINT LEFT$(
MG$(Z),MG(Z))
3690 IF C$="R" THENTL=0:PRINT"New line: ";:GOSUB130:PRINTCHR$(17):
TL=-1:IF CT=0 THEN 3460 ELSE LSET MG$(Z)=I$:MG(Z)=CT:GOTO3460
3700 MG(Z)=0:GOTO3460
```

End

Program Listing 2. BBS module for reading and printing data-base files and messages.

```
350 E$=INKEY$:N=VAL(RIGHT$(F2$,4)):T$=T1$:TT$=F1$:S8$=S1$:S9$=CHR$
(ASC(S2$) AND 15):S7$=CHR$(ASC(S2$) AND 240):S6$=MID$("0123456789A
BCDE",ASC(S9$),1)
360 RD=-1:IF FN P(RN,MN$)<0 THEN RD=0
380 IFINSTR(SE$,S6$)=0THENRD=0
390 IF ((ASC(S7$) AND 32)=32) AND (LEFT$(T$,LEN(NA$))<>NA$) AND (L
EFT$(TT$,LEN(NA$))<>NA$) AND NOTSY THEN RD=0
400 IF (SF$="T" AND INSTR(T$,SS$)=0) OR (SF$="F" AND INSTR(TT$,SS$
)=0) OR (SF$="S" AND INSTR(S8$,SS$)=0) THEN RD=0
410 IF (D$="M") AND (ASC(S7$) AND 16)=16 THEN RD=0
415 IF E$=CHR$(3)THEN RN=E
420 RETURN
4720 GOSUB1230:IF PR THEN 5030
4730 PRINT"Message #";N;" SEC. ";S6$;" ";S$(ASC(S9$))
4740 L=INSTR(T1$,"  "):IF L = 0 THEN L=LEN(T1$)
4750 PRINT"To: ";LEFT$(T1$,L);" ";
4760 IF (ASC(S7$) AND 16)=16 THEN PRINT"(X)";
4770 IF (ASC(S7$) AND 32)=32 THEN PRINT"(P)";
4780 PRINT
4790 L=INSTR(F1$,"  "):IF L=0 THEN L=LEN(F1$)
4800 PRINT"From: ";LEFT$(F1$,L)
4810 L=INSTR(S1$,"  "):IF L=0 THEN L=LEN(S1$)
4820 PRINT"Subject: ";LEFT$(S1$,L)
4830 PRINT"Date/Time: ";T2$:PRINTLEFT$(S$,SV)
4840 IF SF THEN RETURN
4850 OPEN"I",3,F2$+"/BBS"+DD$
4860 GOSUB1230:IF PR THEN 5130
4870 POKE RE,SV
4880 IF EOF(3) THEN 5020
4890 IF INKEY$=CHR$(16) THEN 5020
```

Listing 2 continued

After the caller addresses the message and lists its subject, lines 3350–3380 ask the caller if the header is accurate. If so, the program prompts the caller to enter the message in the allotted space: 20 lines of 64 characters. The first line is indicated by 1:.

Lines 3400–3440 transfer the message from a GOSUB 130 (the telecommunicating Input statement) and LSETs I$ (the characters received) into MG$(LN), a temporary storage array. The program has previously dimensioned each of the elements here to 80 characters so you don't lose any additional string space.

Line 3420 checks for carriage returns (CHR$(141)), removing them if the line isn't indented. Lines 3450–3700 deal with leave options. The GOSUB 860 in line 3540 inserts the message into the string MSG0000/BBS, constructing the secret file that we discussed last month. MN$ is an in-memory index of all the messages on the board. Because each message number is represented by a 2-byte string in MN$, it's easy to locate a message. We'll learn to manipulate MN$ in a later column.

Reading and Printing

Program Listing 2 lets callers read messages on the board. We erroneously omitted lines 350–420 from last month's listings. These lines decode the information bit-mapped into S2$. They also contain the file name and other fields of Messages/BBS, and decide if the caller is allowed to read the message.

Lines 4720–4830 put the header information on the screen, while lines 5030–5110 send this information to the line printer. If the scan flag (SF) in line 4840 is set to SF = −1, it prints only the headers, not the text. This gives you the option of reading or scanning the messages with a single routine.

Lines 4850–5020 are responsible for the text of both the messages and data-base files. Line 4850 opens the secret file on the data drive, while line 4870 POKEs address RE with the video width, starting the reading process. The program tests for end-of-file in line 4880. Line 4890 checks for a control-P (CHR$(16)), indicating that the caller has ceased reading. Line 5020 then brings you to Close and Return.

The exception to the video formatting routine is the .IMG extension. You don't want Basic programs with carriage returns in the middle of them, because the caller's Basic won't load them. The .IMG extension signals the program to forget about video formatting.

Line 4895 checks the UART status for loss-of-carrier if you're not operating from the console. If the caller hasn't hung up without first logging off, line 4900 invokes a LINE INPUT command that puts a string from disk into A$. Line 4910 changes the CHR$(141)s to carriage returns for printing. Line 4920 formats the string to the caller's screen width.

The exception to the video formatting routine is the .IMG extension. For example, you don't want Basic programs (XA files) with carriage returns in the middle of them, because the caller's Basic won't be able to load them; therefore, the .IMG extension on a file signals the program to forget about video formatting.

Finally, the program prints each character in A$, while it checks for control-P, XOFF (CHR$(19)), XON (CHR$(17)), and a loss of carrier.

Next month we'll be searching the message board, so watch for the BBS Express as it picks up steam. ■

The BBS Express, 80 Micro*'s bulletin board system, is open 24 hours a day. Call us at 603-924-6985 to see the finished product. UART parameters are 300 baud, seven bit words, one stop bit, and even parity.*

You can reach J. Stewart Schneider and Charles E. Bowen either through their bulletin board at 606-739-6088 or c/o Saturday Software, P.O. Box 404, Catlettsburg, KY 41129.

Listing 2 continued

```
4895 IF ((INP(&HE8)AND32)=32) AND NOTWZ THEN 2630
4900 LINEINPUT#3,A$
4910 CR=INSTR(A$,CHR$(141)):IFCR>0 THEN MID$(A$,CR,1)=CHR$(13):GOT
O4910
4920 IF INSTR(CM$,".IMG")=0 THEN Z=USR2(VARPTR(A$))
4930 FOR C=1 TO LEN(A$)
4940 PRINT MID$(A$,C,1);
4950 IF INKEY$=CHR$(16) THEN 5020
4960 IF INKEY$<>CHR$(19) THEN 5010
4970 IF INKEY$=CHR$(17) THEN 5010
4980 IF WZ THEN 4970
4990 IF (INP(&H0E8) AND 32)=0 THEN 4970
5000 GOTO2630
5010 NEXT:GOTO4880
5020 PRINT CHR$(17):CLOSE3:PRINT:RETURN
5030 LPRINT"Message #";N;" Sec. ";S6$;" ";S$(VAL(S6$)+1)
5040 LPRINT"To: ";T1$;
5050 IF (ASC(S7$) AND 16)=16 THEN LPRINT"(X)";
5060 IF (ASC(S7$) AND 32)=32 THEN LPRINT"(P)";
5070 LPRINT" "
5080 LPRINT"From: ";F1$
5090 LPRINT"Subject: ";S1$
5100 LPRINT"Date/Time: ";T2$:LPRINT LEFT$(S$,SV)
5110 IF SF THEN RETURN
5120 OPEN"I",3,F2$+"/BBS"+DD$
5130 POKE VW,80:POKE RE,80
5140 IF EOF(3) THEN POKE VW,SV: GOTO 5020
5150 LINEINPUT#3,A$
5160 CR=INSTR(A$,CHR$(141)):IF CR>0 THEN MID$(A$,CR,1)=CHR$(13):GO
TO 5160
5170 Z=USR2(VARPTR(A$))
5180 LPRINT A$;:GOTO 5140
```

End

See List of Advertisers on Page 176

Using an Index For More Flexibility

The message board is the heart of any BBS. Not only is it where callers communicate with one another, it's the sysop's forum for providing information about the BBS. We've concentrated on storing messages for the last few months; now we'll show you how to read and scan the message board.

Your main priority in developing a program to read the message board is flexibility. This month's programs let you read messages forward, backward, individually, selectively, and those marked specifically for your mailbox.

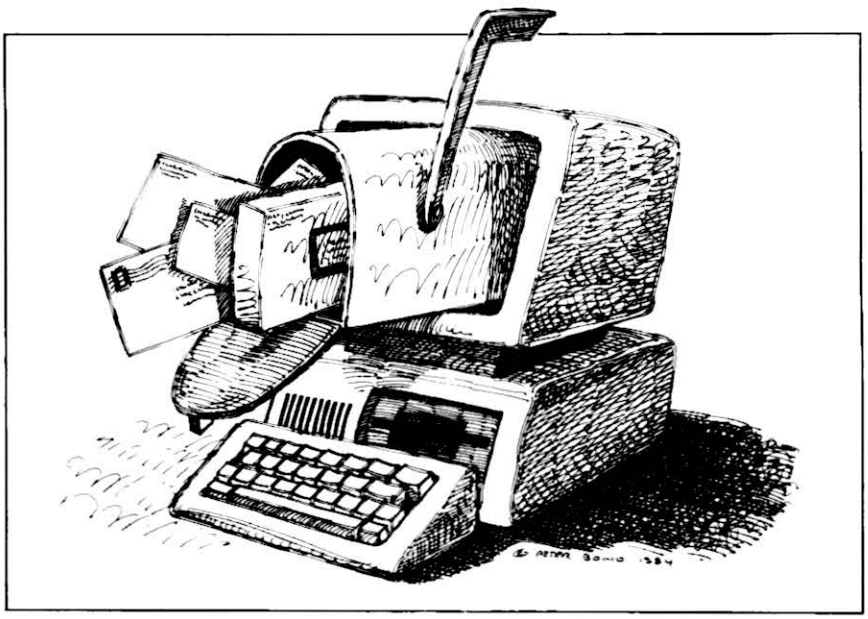


Illustration by Peter Bono

Indexing

Reading messages marked for a mailbox isn't too difficult in terms of programming. When the caller logs on, the BBS Express looks for messages directed to the caller, saves the message's record number in an integer array, and retrieves the message individually in a For...Next loop.

Reading the message board forward, backward, individually, or selectively, however, requires a different technique. The BBS software must find a specific message on the board so that it can go directly to that message, rather than waste time searching through the entire message file (Messages/BBS). You need to develop an index of all the messages that pinpoints the location of a message by record number so that the BBS Express can retrieve it quickly.

The BBS's index holds all the messages' record numbers; it's labeled MN$. Program Listing 1 develops MN$; it's written in machine language because Basic would work too slowly.

MN$ stores a message's record number according to a mathematical formula. You develop the index with the MKI$ command, which does all the record-number calculations for you. By setting MN$ = MKI$(message number), MN$ becomes two characters that represent the record number, in least significant byte/most significant byte format. To maintain the index, MN$ = MN$ + MKI$(message number) is set as users add messages to the board.

This saves you time because you can quickly scan the index for a message number and then get its record number. Once you know the locations and numbers, you can specify the manner in which you want to read the board.

Program Listing 1, the Assembly-language code, locates the message number, while Program Listing 2, in Basic, manipulates this information so that you can read the board in a variety of ways.

How the Index Works

Line 1760 calls PARAM, which returns the starting address for MN$ to the HL register and the length of MN$ to the B register, provided Basic calls this routine with a statement like USR3(VARPTR(MN$)), found in line 430 of Program Listing 2.

While it may seem that line 1770 loads the DE register pair with zero, the Basic program actually POKEs the number of the message you want, in least significant byte/most significant byte format, over the zeros. For example, if you're looking for message 210, line 1770 responds with LD, DE, 210.

The program transfers the index's starting address to the IX register by PUSH HL, POP IX in lines 1780 and 1790. The C register in line 1800 acts

The Key Box

LOAD 80

Model III
48K RAM
Disk Basic
Assembly Language
Two Disk Drives
Editor/Assembler

Program Listing 1. BBS module that creates an index for the message board.

```
01750 ;
01760 FSRCH   CALL    PARAM           ;GET VARPTR(MN$)
01770 SEARCH  LD      DE,0000         ;SEARCH STRING
01780         PUSH    HL
01790         POP     IX
01800         LD      C,255           ;STR. POS. COUNTER
01810 FSR010  LD      L,(IX+0)
01820         LD      H,(IX+1)        ;GET FIRST PAIR FOR CMPR
01830         INC     C
01840         INC     C               ;BUMP STRNG POINTER
01850         BIT     7,H             ;CHECK FOR NEGETIVE
01860         JR      NZ,REVR         ;NEGETIVE - DEAD FILE
01870         RST     18H             ;COMPARE HL/DE
01880         JR      Z,FSR100        ;FOUND
01890         JR      NC,FSR100       ;TARGET>SOURCE
01900 REVR    INC     IX
01910         INC     IX
01920         DEC     B
01930         DJNZ    FSR010          ;LOOP TIL FOUND
01940 FSR100  INC     C               ;STRPOS+1
01950         SRL     C               ;(STRPOS+1)/2
01960         LD      B,0
01970         PUSH    BC
01980         POP     HL
01990         JP      BASIC           ;PASS TO BASIC
02000;
```

End

Program Listing 2. BBS module to read the message board.

```
300 IF LEN(CM$)<=1 THEN C$=CM$:CM$="":RETURN
310 C$=LEFT$(CM$,1):CM$=FNS$(CM$):IF (C$<=CHR$(32)) OR (C$=";") TH
EN 300
320 IF CM$=""THEN RETURN
330 IF (ASC(CM$)<=32) OR (ASC(CM$)=59) THEN CM$=FNS$(CM$):GOTO320
340 RETURN
350 E$=INKEY$:N=VAL(RIGHT$(F2$,4)):T$=T1$:TT$=F1$:S8$=S1$:
S9$=CHR$(ASC(S2$) AND 15):S7$=CHR$(ASC(S2$) AND 240):S6$=MID$("012
3456789ABCDE",ASC(S9$),1)
360 RD=-1:IF FN P(RN,MN$)<0 THEN RD=0
380 IF INSTR(SE$,S6$)=0THEN RD=0
390 IF ((ASC(S7$) AND 32)=32) AND (LEFT$(T$,LEN(NA$))<>NA$) AND (L
EFT$(TT$,LEN(NA$))<>NA$) AND NOTSY THEN RD=0
400 IF (SF$="T" AND INSTR(T$,SS$)=0) OR (SF$="F" AND INSTR(TT$,SS$
)=0) OR (SF$="S" AND INSTR(S8$,SS$)=0) THEN RD=0
410 IF (D$="M") AND (ASC(S7$) AND 16)=16 THEN RD=0
415 IF E$=CHR$(3)THEN RN=E
420 RETURN
430 POKE FD+1,INT(MN/256):POKE FD,MN-(INT(MN/256)*256):S=USR3(VARP
TR(MN$)):RETURN
440 IF (SF) THEN RETURN ELSE RC=0:IF ((LEFT$(T$,5)="SYSOP") AND SY
) OR (LEFT$(T$,LEN(NA$))=NA$) THEN RC=-1
450 K=RC:IF (LEFT$(TT$,LEN(NA$))=NA$) OR (SY) THEN K=-1
460 IF RC THEN S7$=CHR$(ASC(S7$) OR 16):LSET S2$=CHR$(ASC(S7$) OR
ASC(S9$)):PUT 1,RN
465 IF (BM) OR (PR) THEN RETURN
470 PRINT"(C RE T)";:IF K THEN PRINT"(D = DELETE)";
480 IF SY THEN PRINT"(P = PRINT IT)";
490 GOSUB130:PRINTCHR$(17);
500 IF SY AND ASC(I$)=80 THEN PR=-1:GOSUB4720:PR=0:GOTO470
510 IF LEFT$(I$,1)="T" THEN RN=E:RETURN
520 IF LEFT$(I$,2)="RE" THEN 3380
530 IF NOT((LEFT$(I$,1)="D") AND K) THEN 545
540 PRINT"Please confirm delete (y/n)";:GOSUB130:PRINTCHR$(17):IF
ASC(I$)=89 THEN MID$(MN$,2*RN-1,2)=MKI$(-FNP(RN,MN$)):
PRINT"Deleted":MID$(I$,1,1)="C"
545 IF LEFT$(I$,1)="C"THEN RETURN
550 PRINT"C = Continue":PRINT"RE = REply to message":PRINT"T = Top
 (Exit Read Function)":IF K THEN PRINT"D = Delete message"
560 GOTO470
800 PRINT"Section (Tap ENTER for all, ? for list)";:GOSUB130:PRINT
CHR$(17);
810 IF CT=0 THEN SE$=UA$:RETURN
820 IF ASC(I$)=63 THEN GOSUB830:GOTO800
821 SE$=LEFT$(I$,CT):FOR Z=1TOCT:IF INSTR(UA$,MID$(SE$,Z,1))=0THEN
PRINT"Unauthorized to section ";MID$(SE$,Z,1):GOTO800 ELSE NEXT:RE
TURN
```

Listing 2 continued

You can read through the message board in six ways, and you can chain the commands to save time.

as a counter, and starts out at 255. Line 1810 loads the HL register with the first two bytes of MN$, representing the number of the message in the first position. The program then increments C twice; the first increment moves C from 255 to zero, the second from zero to 1.

Line 1850 uses the Bit command to find out if bit 7 of the message number is set. If it is, it's a negative number, which means that the BBS has deleted the message. The program ignores this and jumps to line 1900. If the number is positive, it must be compared with the number in the DE register. Line 1870 calls a Restart command, RST 18H, which compares the HL and DE registers, then sets the proper flags.

If the Restart sets the Z flag, it has found the target message number and the program jumps to FSR100. If the carry flag isn't set, you've passed the target, and the program again jumps to FSR100. Otherwise, you bump IX twice to point to the next message number, decrement B, and loop until the program finds the number.

FSR100 bumps the character counter, register C, by 1, then shifts one position to the right and divides by 2, loading the B register with a zero. Lines 1970 and 1980 transfer this to the HL register and pass the value of the target message on to the Basic code in Program Listing 2. If the target message doesn't exist, the next-largest value is returned to Basic.

Reading Messages

Program Listing 2 lets you read through the message board in six ways: forward (the RF command), in reverse (RR), individually (RI), marked messages (RM), new messages (RN), and selectively (RS).

You can chain these commands, too. This is a real time-saver; for example, if you want to read from message 210 on, issue the R command and wait to be prompted for the direction

and message number. If you prefer to skip the prompts, just type in RF210.

CM$, in line 2870, contains whatever the caller entered at the command prompt. If CM$ is a null string, the program prompts for a subcommand. Otherwise, GOSUB 300 transfers the second character of CM$ to D$.

Line 2900 checks to see if D$ is an authorized command. If it's not, an authorized command summary is printed. If, for example, D$ is N, the caller has asked to read new messages. The program then sets MN, the starting message number, to HM + 1 (HM is the caller's high message number on his last visit). D$ becomes F because you read new messages forward.

CM$ still contains the message number, which line 2940 sets as MN = VAL(CM$). If the value is less than one, the program prompts for the starting message. Otherwise, it jumps to line 2970. Line 2970 sets SE$ to UA$, the user's authorized access, while GOSUB 200 opens Messages/BBS. Unless a caller requests to read individually or marked, the GOSUB 800 prompts for the section numbers to read and validate the caller's choice.

Line 2980 directs each command to its proper routine. Line 2990 is the Read Forward section where the GOSUB 430 sets S equal to the record number containing the requested starting message. E is set to SN, the number of messages on the system.

Line 3000 gets each record in Messages/BBS, starting with Record S. The GOSUB 350 initiates a routine to check whether or not the caller has access to the message. If the message is private, the program prints a period and checks the next message. If the caller does have access to the message, GOSUB 4720 prints it.

If the requested message number is greater than HI, the highest message number on the caller's last visit, HI becomes the present message number. The BBS writes this new HI to disk when the caller closes out.

GOSUB 440 prints four options: continue with the C command, return to the top menu (T), delete the message (if he's the sysop, sender, or addressee), or print the message (if he's the sysop). These closing prompts can be eliminated with brief mode (BM = −1), or if the SYSOP calls for a hard copy (PR = −1).

All of the Read commands are built on the same structure. The one exception is Read Individual, where the exact message must be specified. Line 3090 checks this; if it's not MN, the message doesn't exist on the system. Line 2860 sets SF = −1, which lets you scan the message board. ■

The BBS Express, 80 Micro*'s bulletin board system, is open 24 hours a day. Call us at 603-924-6985 to see the finished product. UART parameters are 300 baud, 7-bit words, one stop bit, and even parity.*

You can reach J. Stewart Schneider and Charles E. Bowen either through their bulletin board at 606-739-6088 or c/o Saturday Software, P.O. Box 404, Catlettsburg, KY 41129.

✓ See List of Advertisers on Page 192

Listing 2 continued

```
830 FOR Z=1 TO 15:IF (S$(Z)=STRING$(16,32)) OR (INSTR(UA$,MID$(SQ$
,Z,1))=0) THEN 850
840 PRINT MID$(SQ$,Z,1);" ";S$(Z)
850 NEXT:RETURN
2850 CLS:PRINTCHR$(12);"Scanning Message Board"
2860 PRINT"Enter a Control-C to stop.":SF=-1
2870 IFCM$<>""THEN2890
2880 PRINT"SUBCOMMAND (? For HELP): ";:GOSUB130:PRINTCHR$(17):IF C
T=0 THEN 1720 ELSE CM$=LEFT$(I$,CT)
2890 GOSUB300:D$=C$:IF D$="M" THEN 2970
2900 IF INSTR("FRISMN",D$)=0 THEN D$="?"
2910 IF D$="?" THEN PRINT"? - Prints this list":PRINT"F - Forward"
:PRINT"R - Reverse"
2920 IF D$="?" THEN PRINT"I - Individual":PRINT"S - Selected":PRIN
T"M - Marked":PRINT"N - New Messages":GOTO2880
2930 IF D$="N" THEN MN=HM+1:D$="F":GOTO2970
2940 MN=VAL(CM$):IF MN>0 THEN 2970
2950 PRINT"System contains messages"SL"to"SH:IF D$="I" THEN PRINT"
Read which message?"; ELSE PRINT"Read starting with which message?
"
2960 GOSUB130:PRINTCHR$(17):MN=VAL(I$)
2970 SE$=UA$:GOSUB220:IF D$<>"I" AND D$<>"M" THEN GOSUB800
2980 ON INSTR("FRISM",D$) GOTO2990,3040,3090,3130,3220
2990 E=SN:GOSUB430
3000 FOR RN=S TO E:GET 1,RN:GOSUB350:IF NOT RD THEN PRINT".";:GOTO
3020
3010 PRINT:GOSUB4720:GOSUB440:IF N>HI THEN HI=N
3020 IF INKEY$=CHR$(3) THEN RN=E
3030 NEXT:GOTO1720
3040 E=1:GOSUB430
3050 FOR RN=S TO E STEP-1:GET 1,RN:GOSUB350:IF NOT RD THEN PRINT".
";:GOTO3070
3060 PRINT:GOSUB4720:GOSUB440:IF N>HI THEN HI=N
3070 IF INKEY$=CHR$(3) THEN RN=E
3080 NEXT:GOTO1720
3090 GOSUB430:IF FN P(S,MN$)<>MN THEN PRINT"NO SUCH MESSAGE":GOTO1
720
3100 RN=S:GET 1,RN:GOSUB350
3110 IF RD THEN GOSUB4720:GOSUB440:IF N>HI THEN HI=N
3120 GOTO1720
3130 PRINT"Search field:(T,F,S)";:GOSUB130:PRINTCHR$(17):IF CT=0 T
HEN 1720 ELSE C$="F"
3140 SF$=LEFT$(I$,1):IF INSTR("TFS",SF$)=0 THEN 3130
3150 PRINT"Search for: ";:GOSUB130:PRINTCHR$(17):
SS$=LEFT$(I$,CT):PRINT"Searching";3160 E=SN:GOSUB430
3170 FOR RN=S TO E:GET 1,RN:GOSUB350:IF NOT RD THEN PRINT".";:GOTO
3200
3180 PRINT:GOSUB4720:GOSUB440:IF N>HI THEN HI=N
3190 PRINT".";
3200 IF INKEY$=CHR$(3) THEN RN=E
3210 NEXT:SF$="":SS$="":GOTO1720
3220 E=PM:IF E=0 THEN PRINT"No Marked Messages":GOTO1720
3230 FOR X=1 TO E:RN=M(X):GET 1,RN:GOSUB350
3240 GOSUB4720:GOSUB440:IF N>HI THEN HI=N
3250 IF LEFT$(I$,1)="T" THEN X=E
3260 IF INKEY$=CHR$(3) THEN X=E
3270 NEXT:GOTO1720
```

End

Getting the Message Across

One of the most important functions of a BBS is its ability to search and find information. Successful BBSes can locate information accurately and efficiently. This month's BBS Express concentrates on finding messages both in memory and on disk.

While it's faster to locate information in memory than on disk, memory space is more limited, which in turn limits the amount of information you can keep in memory. A message header, for example, consists of the message number, subject, date, time, section number, name of sender, and destination. Ideally, all this information should be in memory so you can locate a message quickly. This information requires a total of 92 bytes for each header, or 9.2K of memory if there are 100 messages on board. Unfortunately, however, that's a lot of memory.

Illustration by Roy Lewando

To complicate matters further, this header information is in string form, which can easily trigger string garbage collection. The message number, however, isn't in string form. An acceptable solution, then, is to create a fast, in-memory search for the message number, leaving all other searches as disk operations. Because the message numbers are stored economically, memory isn't heavily taxed.

Storing the Message Number

Your computer's memory stores integers in Z80 least significant byte/most significant byte (LSB/MSB) format. Each integer requires 2 bytes: The first byte contains the least significant byte, while the second byte contains the most significant byte. In message number 256, for example, the least significant byte is 0, and the most significant byte is 1. The VARPTR command returns the address where the BBS stores the least significant byte; it stores the most significant byte in the next position.

Using this information, you build an in-memory index of the message numbers that you access with a short machine-language code (see Program Listing 1). The index is simply a string that contains the message numbers in sequential order, making it easy to search for any given message number. The VARPTR (string variable) returns the memory address of a pointer that contains both the length of the string and the memory address of the string's first character. Because this information is all part of the same string, it's easily written to disk.

Now you need to convert the message numbers to and from the Z80 LSB/MSB format. Disk Basic does this: It contains MKI$, which returns a 2-byte string in LSB/MSB format, and CVI, which takes that string and returns a number. In the case of message number 256, MKI$(256) = CHR$(0) + CHR$(1); CVI(CHR$ (0) + CHR$ (1)) = 256. Each time a user leaves a message, the BBS writes MKI$ (message number) to the in-memory string. The BBS stores the header for the Nth message in record N, and finds it in the index string at position 2*N – 1. If P is the position in the string, and RN is the record number of the header for the message, then

$$RN = (P + 1)/2$$
$$P = (2*RN) - 1$$

Program Listing 1, in Assembly language, searches the index for the message number. With this message number, you have access to the header, which gives you the secret file name of the text of the message. Label FSRCH calls the PARAM routine which places the length of the string passed from Basic in the B register, and the address of the string's first character in the HL register pair. Basic POKEs the message number that you're looking for in SEARCH + 1.

The Key Box

LOAD 80

Model III
48K RAM
Disk Basic
Assembly Language
Two Disk Drives
Editor/Assembler

The BBS then loads this value into the DE pair, transfers the address of the first character in the string to the IX register with the PUSH HL, POP IX instructions, and loads C with 255 to act as a counter.

The BBS loads HL with the value that's located in the first two positions of the string. C is bumped twice, changing its value to 1. Because deleted messages are marked by being set to a negative value (your computer uses 2's complement notation), if bit 7 of the value in the H register is a 1, it's negative and thus ignored as a killed record.

If the number in this position isn't negative, it's compared with the value that we're searching for by the RST 18H instructions. In effect, RST 18H, in 1 byte, subtracts the contents of the DE register pair from the contents of the HL register pair, ignoring the result yet saving the flag status.

If the message numbers exceed the limit, the BBS removes all deleted messages and text files.

If the RST 18H command sets the zero flag, the BBS has found the message number. But if the Carry flag isn't set, you've gone too far. In either event, the BBS increments the C counter by 1, then divides it by 2 by shifting it one position to the right (SRL C), placing zero in the B register, and moving BC to HL to pass to Basic.

If the message number is still unknown, the BBS bumps IX, to point to the next pair of characters, decrements B, and initiates DJNZ, looping to FSR010 until done. The JP BASIC instruction passes the value in the HL register (the record number that we're seeking) to Basic.

Program Listing 2 shows how Basic interfaces with the code in Program Listing 1. Basic defines FD as the address of SEARCH+1. Line 430 POKEs the LSB of the number that you're searching for into SEARCH+1, and the MSB into SEARCH+2. A call to USR3 returns S equal to either the record number you want or to the next-highest record number.

Lines 2730–2840 make up the close-out routine. This is done first by compressing the number of messages on the board so they don't exceed the maximum set by the sysop. If you're under the limit, write the index back to disk. If, however, the message numbers exceed the limit, the BBS removes all deleted messages and text files. Any excess messages are removed with NW$, a new index string. This new index starts removing messages at the high end of MN$ until NW$ is equal to 2*MX, the maximum number of messages the sysop allows. Everything below that point gets killed, and NW$ is written to disk. ■

Program Listing 1. BBS code that locates message numbers.

```
01740 ;         SEARCH FOREWARD FOR SPECIFIED 2-BYTE STRING
01750 ;
01760 FSRCH     CALL    PARAM           ;GET VARPTR(MN$)
01770 SEARCH    LD      DE,0000         ;SEARCH STRING
01780           PUSH    HL
01790           POP     IX
01800           LD      C,255           ;STR. POS. COUNTER
01810 FSR010    LD      L,(IX+0)
01820           LD      H,(IX+1)        ;GET FIRST PAIR FOR
CMPR
01830           INC     C
01840           INC     C               ;BUMP STRNG POINTER
01850           BIT     7,H             ;CHECK FOR NEGETIVE
01860           JR      NZ,REVR         ;NEGETIVE - DEAD FILE
01870           RST     18H             ;COMPARE HL/DE
01880           JR      Z,FSR100        ;FOUND
01890           JR      NC,FSR100       ;TARGET>SOURCE
01900 REVR      INC     IX
01910           INC     IX
01920           DEC     B
01930           DJNZ    FSR010          ;LOOP TIL FOUND
01940 FSR100    INC     C               ;STRPOS+1
01950           SRL     C               ;(STRPOS+1)/2
01960           LD      B,0
01970           PUSH    BC
01980           POP     HL
01990           JP      BASIC           ;PASS TO BASIC
```

End

Program Listing 2. BBS code to interface Program Listing 1 with Basic.

```
430 POKE FD+1,INT(MN/256):POKE FD,MN-(INT(MN/256)*256):
S=USR3(VARPTR(MN$)):RETURN
2730 IF (SN=0) OR (LEN(MN$)=0) THEN SL=0:SH=0:SN=0:NW$="":
GOTO 2820
2740 FOR S=LEN(MN$)-1 TO 1 STEP-2:MN=CVI(MID$(MN$,S,2))
2750 IF (LEN(NW$)<2*MX) AND (MN>0) THEN
NW$=MID$(MN$,S,2)+NW$:P$=CHR$(INT((S+1)/2))+P$:GOTO2770
2760 A$="MSG0000/BBS"+DD$:GOSUB860:KILL A$:SN=SN-1
2770 NEXT:IF LEN(NW$)>1 THENSL=CVI(LEFT$(NW$,2)):
SH=CVI(RIGHT$(NW$,2)):GOSUB220 ELSE SL=0:SH=0:GOTO 2820
2780 FOR P=1 TO SN
2790 GET 1,ASC(MID$(P$,P,1))
2800 PUT 1,P
2810 NEXT:CLOSE
2820 GOSUB190:GET 3,1:LSET SL$=MKI$(SL):LSET
SH$=MKI$(SH):LSET SN$=MKI$(SN):LSET SC$=MKI$(SC+1):LSET
ND$=MKI$(ND):LSET NM$=MKI$(NM):LSETDS$=MKI$(DS):PUT 3,1
2830 FIELD 3,255 AS ZM$:LSET ZM$=NW$:PUT 3,3:CLOSE
2840 RUN
```

End

The BBS Express, 80 Micro*'s bulletin board system, is open 24 hours a day. Call us at 603-924-6985 to see the finished product. UART parameters are 300 baud, 7-bit words, one stop bit, and even parity.*

You can reach J. Stewart Schneider and Charles E. Bowen either through their bulletin board at 606-739-6088 or c/o Saturday Software, P.O. Box 404, Catlettsburg, KY 41129.

Branching Out With Your BBS

Computer bulletin board systems (BBSes) often need to search for information, and the BBS Express is no exception. The BBS searches two lists constantly—the membership log and the data base catalog. Searching lists like these is potentially messy when you consider that they're often long and randomly compiled. Common search methods, such as sorting and sequential searching, aren't efficient because of the random nature of our lists. Using a binary tree, however, is an efficient and conveniently applied method that allows easy access to random information.

This month, we'll discuss the binary tree as it applies exclusively to the membership log. This is an ideal place for a binary tree because the more random the information supplied, the more efficient the binary tree. When employed here, the binary tree produces a sorted list when needed, and finds an entry in a 256-record file with surprisingly few disk accesses.

The example listing in Fig. 1 sets up a random-access file that lets the caller enter his first name. Two additional fields are added to the membership list: a left pointer (LP$) and a right pointer (RP$). The GOSUB 870 in line 90 manipulates the two pointers in a way that lets you quickly find a name.

When a user enters a new name on the binary tree, the BBS compares it to the name at the first junction where the tree splits into its left and right branches (see Fig. 2). If the new name is lower in the alphabet than the name it's compared with, it goes to the left branch. Otherwise, it goes to the right branch. The same comparison is made at the next junction, where again the lower name goes to the left. If at the end of the branch, there are no more junctions, the name is added.

At each junction, the remaining items are divided in half. If the tree is balanced, half of the remaining list lies on the left path, the other half on the right. By taking the left or right branch, you effectively ignore half the remaining list. Therefore, if you start with 256 entries on the tree, 128 entries are eliminated after the first comparison. With each respective comparison, you reduce this figure to 64, 32, 16, eight, four, two, and eventually, one. With the binary tree method, we're able to locate an item in just nine comparisons.

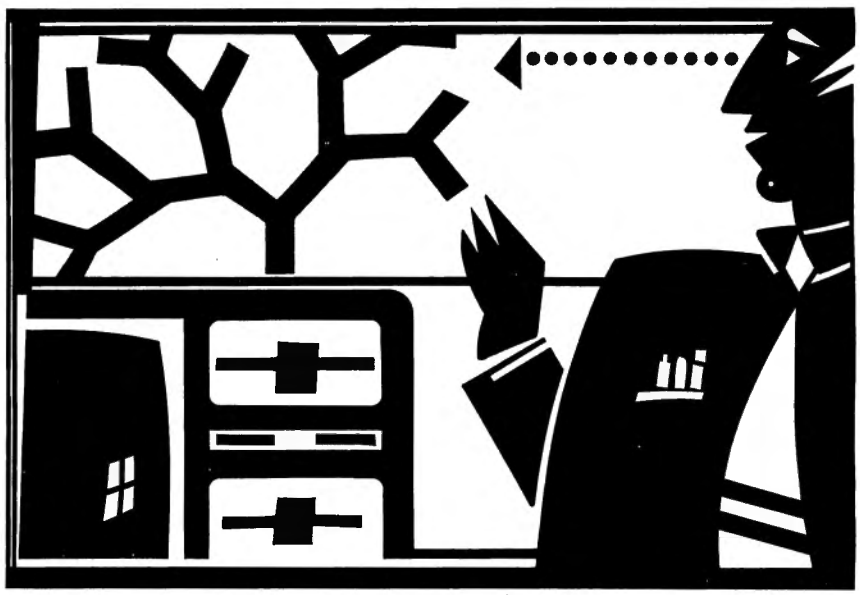


Illustration by Phil Geraci

Proper Coding

Because the records on a disk are lined up one after another, you're able to arrange the pointers so that they contain an offset from the present record to the next smaller record (for LP$), or the next larger record (for RP$). If, for example, you're at record 1, and the next record alphabetically smaller than the name in record 1 is in record 5, and the next larger is record 4, LP$ will contain 4, and RP$ will contain 3. When the pointer you want to move is a zero, you've reached the end of a branch, and the BBS adds the name.

Line 870 of both Fig. 1 and the Program Listing tests for EN=1. If there's only one record in the file, the program exits the search with a return. Otherwise, the BBS initializes record Y to 1.

Line 890 compares CK$, the name that you want to position, with N1$, the name from record Y. If CK$ is greater than NI$, the BBS sets FS equal to the value of the right pointer. CVI simply converts the string representation of a number in a random file to a number.

If this right pointer is zero, it's set to the difference between record Y and EN, the record written to disk by line 80 of Fig. 1. Then the BBS unites record Y with the updated pointer before it exits the program. If CK$ is less than or equal to the name in record Y, the BBS repeats the procedure with the left pointer, LP$. Otherwise, you

The Key Box LOAD 80

Model III
48K RAM
Disk Basic
Two Disk Drives

```
10 CLS:CLEAR1000:NM=1:CK$=STRING$(20,32)
20 OPEN"R",2,"TEST/DAT:0"
30 FIELD 2, 20 AS N1$,2 AS LP$,2 AS RP$
40 INPUT"ENTER NAME, QUIT TO STOP";N$
50 IF N$="QUIT" THEN CLOSE:END
60 LSET N1$=N$:LSET LP$=MKI$(0): LSET RP$=MKI$(0)
70 NM=NM+1
80 PUT 2,NM
90 EN=NM:LSET CK$=N$:GOSUB 870
100 GOTO 40
870 Y=1:IFEN=1 THEN RETURN
880 GET 2,Y
890 IF CK$>N1$ THEN FS=CVI(RP$):IF FS=0 THEN LSET
RP$=MKI$(EN-Y):PUT 2,Y:RETURN
900 IF CK$<=N1$ THEN FS=CVI(LP$):IF FS=0 THEN LSET
LP$=MKI$(EN-Y):PUT 2,Y:RETURN
910 Y=Y+FS:GOTO880
```

Figure 1. Sample code for binary tree sort.

```
10 CLS:CLEAR1000:EN=256:CK$=STRING$(20,32)
20 OPEN"R",2,"TEST/DAT:0"
30 FIELD 2, 20 AS N1$,2 AS LP$,2 AS RP$
40 INPUT"ENTER NAME, QUIT TO STOP";N$
50 IF N$="QUIT" THEN CLOSE:END
60 LSET CK$=N$:GOSUB 940
70 IF ER THEN ?"NO SUCH NAME ON THE LIST"
80 IF NOTER THEN PRINT "NAME FOUND IN RECORD";MR
90 GOTO 40
940 IF EN=0 THEN 1010
950 MR=1:ER=0
960 GET 2,MR
970 IF CK$<N1$ THEN FS=CVI(LP$):GOTO1000
980 IF CK$>N1$ THEN FS=CVI(RP$):GOTO1000
990 RETURN
1000 MR=MR+FS:IF FS>0 THEN 960  '  REENTER HERE
1010 ER=-1:RETURN
```

Figure 3. Sample code to search the membership log.

```
10 CLS:CLEAR1000:EN=256:SR$=STRING$(255,32)
20 OPEN"R",2,"TEST/DAT:0"
30 FIELD 2, 20 AS N1$,2 AS LP$,2 AS RP$
40 GOSUB 1020
50 PRINT N1$
60 IF NOTER THEN GOSUB 1080:GOTO50
70 CLOSE:END
1020 PRINT"Sorting...":MR=1:ER=0:Z=1:FS=1:LSET SR$=CHR$(0)
1030 IF DS=0 THEN ER=-1:CLOSE:RETURN
1040 IF FS>0 THEN GET
2,MR:Z=Z+1:MID$(SR$,Z,1)=CHR$(MR):FS=CVI(LP$):MR=MR+FS:GOTO1
040
1050 MR=ASC(MID$(SR$,Z,1)):Z=Z-1:IF MR=0 THEN
ER=-1:CLOSE:RETURN
1060 GET 2,MR:IF LEFT$(P1$,1)=CHR$(0) THEN 1080
1070 RETURN
1080 FS=CVI(RP$):MR=MR+FS:GOTO1040 '  REENTER HERE
```

Figure 4. Sample code to produce a sorted list.

Program Listing. BBS module that creates the membership log binary tree.

```
870 Y=1:IFEN=1 THEN RETURN
880 GET 2,Y
890 IF CK$>N1$ THEN FS=CVI(RP$):IF FS=0 THEN LSET
RP$=MKI$(EN-Y):PUT 2,Y:RETURN
900 IF CK$<=N1$ THEN FS=CVI(LP$):IF FS=0 THEN LSET
LP$=MKI$(EN-Y):PUT 2,Y:RETURN
910 Y=Y+FS:GOTO880
940 IF EN=0 THEN 1010
950 MR=1:ER=0
960 GET 2,MR
```

Listing continued

make Y equal to Y plus FS, which is the offset for either the left branch or the right branch, and then loop to line 880 to repeat the process.

The program code in Fig. 3 lets a caller retrieve a name from the BBS. Lines 10–50 open the file, and let the caller input the name for which he's looking. Line 60 LSETs that name into CK$ before jumping to the routine that starts at line 940. EN has a value of 256, assuming that the file contains 256 records. The BBS initializes MR, the record number, to 1, while it sets ER, the flag that tells you if the record is found, to zero. If the name from record MR is less than CK$, line 970 sets FS to CVI(LP$). If greater, it's sent to the right branch in line 980.

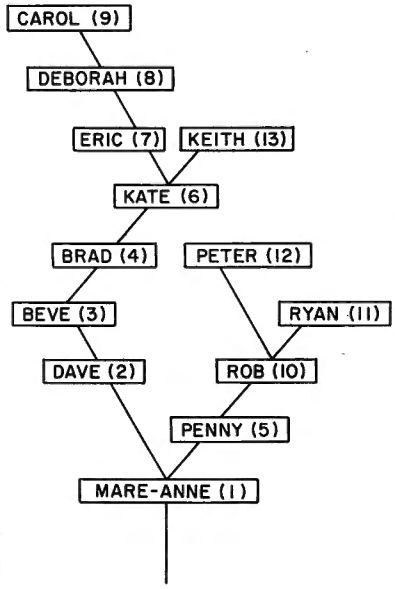


Figure 2. A graphic representation of a binary tree sort. The numbers refer to the order in which the user entered the names.

Line 1000 adds the value of FS to MR. If FS is greater than zero, the routine loops back to line 960 to check the next record. If however, FS is zero, the BBS sets ER to −1 and informs the calling routine that it hasn't found the name.

A Sorted List

The example listing in Fig. 4 produces a sorted list from the binary tree. Line 1040 follows the left pointers to find a zero, creating a path as it moves along. This path, SR$, is a string of 255 blanks, with its first char-

The program code lets a caller retrieve a name from the BBS.

Listing continued

```
970 IF CK$<N1$ THEN FS=CVI(LP$):GOTO1000
980 IF CK$>N1$ THEN FS=CVI(RP$):GOTO1000
990 RETURN
1000 MR=MR+FS:IF FS>0 THEN 960  '  REENTER HERE
1010 ER=-1:RETURN
1020 PRINT"Sorting...":MR=1:ER=0:Z=1:FS=1:LSET SR$=CHR$(0)
1030 IF DS=0 THEN ER=-1:CLOSE:RETURN
1040 IF FS>0 THEN GET
2,MR:Z=Z+1:MID$(SR$,Z,1)=CHR$(MR):FS=CVI(LP$):MR=MR+FS:GOTO1
040
1050 MR=ASC(MID$(SR$,Z,1)):Z=Z-1:IF MR=0 THEN
ER=-1:CLOSE:RETURN
1060 GET 2,MR:IF LEFT$(P1$,1)=CHR$(0) THEN 1080
1070 RETURN
1080 FS=CVI(RP$):MR=MR+FS:GOTO1040 '  REENTER HERE
```

End

acter set to CHR$(0) by the LSET command in line 1020. The entire list has been printed when the program backs out of SR to find the CHR$(0) at the end. The DS in line 1030 of the listing keeps track of the current position in SR for the BBS, though not for the example in Fig. 4.

When line 1040 finds zero, indicating the lowest item on the list, MR is set to the ASCII value of the Zth element of SR in line 1050. If MR is zero, you've printed the entire list. The test for P1$ = CHR$(0) in line 1060 is part of the module in this month's listing, though it's not part of the sample routine. From this point on, the calling routine in line 1080 loops to line 1040 until you print the entire list.

Because your membership log will most likely contain more than 255 names, no attempt is made in the actual coding to produce a membership log sorted by first names only.

For a further discussion on binary tree operation, we recommend Ken Knecht's article, "Plant a Binary Tree" (*80 Micro*, November 1982, p. 242). ■

The BBS Express, 80 Micro*'s bulletin board system, is open 24 hours a day. Call us at 603-924-6985 to see the finished product. UART parameters are 300 baud, seven bit words, one stop bit, and even parity.*

You can reach J. Stewart Schneider and Charles E. Bowen either through their bulletin board at 606-739-6088 or c/o Saturday Software, P.O. Box 404, Catlettsburg, KY 41129.

↙ See List of Advertisers on Page 128

All Systems Go For the BBS Express

With spring just around the corner, we figured we'd do a little spring cleaning for the BBS Express. This means taking care of some unfinished business.

Out first task is to publish the code you need to get your board up and running. The Program Listing gives you this code, while the Table outlines the functions of the individual lines.

Once you type in the listing, your board will be in working order: It welcomes, registers, and updates each user's record. In addition, it will display options in each of its menus that, in turn, direct callers to the proper section of the BBS.

Still missing from the BBS Express is the code for the data base catalog, which we'll include in next month's column. ■

Illustration by Roy Lewando

Program Listing. BBS module.

```
1 REM            Towne Crier v. 2.0
2 REM            version date 5/19/13
3 REM          COPYRIGHT 1984 SCHNEIDER AND BOWEN
10 CLS:CLEAR5000:DEFINT A-Z:I$=STRING$(255,32):SR$=I$:LSETSR$=CH
R$(0):8$=STRING$(62,"-"):CK$=LEFT$(I$,20)
11 CH=(PEEK(-1)=1)
15 DIMI$,CT,CM$,Z,MR,X,A$,P1$,TL,N1$,E,ND,CK$,EN,RN,C$,ER,MN,NM,
F2$,S,SN,SY,A1,LN,LP$,MN$,RD,RP$,S7$,PL$,NA$,SV,AS,C1$,D$,FS
16DIMNC,PR,B$,HI,M,NV,SF$,SH,A2$,AC,M1$,N,SF,T1$,VW,Y,C2$,CC1,CR
,D1$,DF,F1,L,L1$,N5$,ND$,S1$,S2$,TT$,UA$,V1$,CA,DC$,NM$
17 DIMPF,PW$,S6$,S9$,SA$,SE$,SL,SL$,SM$,SP$,SV$,XA$,AD$,CT$,IN$,
K,RE,SB$,SC$,S$,SN$,SR$,T$,T2$,UH$,WZ,XL$,CA$,CV,M2,MX
18DIMNW$,OT$,P,P$,PM,RC,S$,SC,SS$,TN,U1$,U2$,U3$,U4$,UL$,B,C,CL$
,FD,HM,N$,P,TR,TT,U$,ZM$,A,DU$,G,L1,R,X$,Z9,DD$,SD$
20  DIM M(50),MG$(20),MG(20),F$(16),S$(16):DEFUSR0=&HFE00:DEFUSR
1=&HFE6C:DEFUSR2=&HFE7F:DEFUSR3=&HFEF1:TL=-1:NC=0:U$="%
        % ##### ACCESS:###"
30 PP=&HFF2A:POKE PP,255'      PRINTER FILTER SWITCH
40 MN=&HFE04:POKE MN,245'      MAX. RECD LINE LENGTH
50 FD=&HFEF5'                  SEARCH TARGET POKE
60 TT=&HFE45:TN=&HFE10:POKE TT,&H20:POKE TT+1,&H14:POKE TN,&H20:
POKETN+1,&H49'        TONE TEST SWITCHES
70 NP=&HFE2E:POKE NP,&H33'     NO PRINT SWITCH
80 RE=&HFEDD'                   CHAR. REMAINING ON VID LINE
90 VW=&HFEE0'                   VIDEO WIDTH
100 DEF FN FL$(X)=RIGHT$(STR$(X),LEN(STR$(X))-1):DD$=":1":SD$=":
0"
110 DEF FN P(R,A$)=CVI(MID$(A$,2*R-1,2))
```

Listing continued

The BBS Express, 80 Micro*'s bulletin board system, is open 24 hours a day. Call us at 603-924-6985 to see the finished product. UART parameters are 300 baud, 7-bit words, one stop bit, and even parity.*

You can reach J. Stewart Schneider and Charles E. Bowen either through their bulletin board at 606-739-6088 or c/o Saturday Software, P.O. Box 404, Catlettsburg, KY 41129.

System Requirements

Model III LOAD 80
48K RAM
Disk Basic
Two disk drives

Lines	Description
1–120	Defines variables and POKE locations, and dimensions arrays
130–180	Telecommunications input statement
300–340	Command processor
440–560	Handles message prompts
570–790	Alters user records
800–850	Section prompt and display
860	Creates file name and message number from A$
870–910	Locates records in membership log and data base catalog binary tree
820–930	Writes new users to disk
1090–1170	Collects new user information
1235	Calculates time
1240–1355	Retrieves system information from disk, prints title page, waits for caller or console command
1380–1620	Caller log-in routine
1630–1670	Checks caller's mailbox
1680–1730	Welcomes visitor, prints bulletin and main command prompt
1740–1890	Main command functions
1900–2620	Main SYSOP functions
2630–2720	Sign-off routine
2850–3270	Read and scan routines
5190	Error trap

Table. Line definitions for BBS module.

Listing continued

```
120 GOTO1240
130 LSET I$=" "
140 CT=USR0(VARPTR(I$)):IF CT=-1 THEN 2630
150 CR=INSTR(I$,CHR$(13)):IF CR=0 THEN 170
160 IF NC THEN MID$(I$,CR,1)=CHR$(141) ELSECT=CR-1:MID$(I$,CR,1)
=CHR$(32)
170 IF TL THEN z=USR1(VARPTR(I$))
180 RETURN
300 IF LEN(CM$)<=1 THEN C$=CM$:CM$="":RETURN
310 C$=LEFT$(CM$,1):CM$=RIGHT$(CM$,LEN(CM$)-1):IF (C$<=CHR$(32))
 OR(C$=";") THEN 300
320 IF CM$=""THEN RETURN
330 IF (ASC(CM$)<=32) OR (ASC(CM$)=59) THENCM$=RIGHT$(CM$,LEN(CM
$)-1):GOTO320
340 RETURN
440 IF (SF) THEN RETURN ELSE RC=0:IF ((LEFT$(T$,5)="SYSOP") AND
SY) OR(LEFT$(T$,LEN(NA$))=NA$) THEN RC=-1
450 K=RC:IF (LEFT$(TT$,LEN(NA$))=NA$) OR (SY) THEN K=-1
460 IF RC THEN S7$=CHR$(ASC(S7$) OR 16):LSET S2$=CHR$(ASC(S7$) O
RASC(S9$)):PUT 1,RN
465 IF (BM) OR (PR) THEN RETURN
470 PRINT"(C RE T)";:IF K THEN PRINT"(D = DELETE)";
480 IF SY THEN PRINT"(P = PRINT IT)";
490 GOSUB130:PRINTCHR$(17);
500 IF SY AND ASC(I$)=80 THEN PR=-1:GOSUB4720:PR=0:GOTO470
510 IF LEFT$(I$,1)="T" THEN RN=B:RETURN
520 IF LEFT$(I$,2)="RE" THEN GOTO3380
530 IF NOT((LEFT$(I$,1)="D") AND K) THEN 545
540 PRINT"Please confirm delete (y/n)";:GOSUB130:PRINTCHR$(17):I
F ASC(I$)=89 THEN MID$(MN$,2*RN-1,2)=MKI$(-FNP(RN,MN$)):PRINT"De
leted":MID$(I$,1,1)="C"
545 IF LEFT$(I$,1)="C"THEN RETURN
550 PRINT"C = Continue":PRINT"RE = REply to message":PRINT"T = T
op (ExitRead Function)":IF K THEN PRINT"D = Delete message"
560 GOTO470
570 GET 2,MR:PRINTCHR$(12);
580 PRINT"Name: ";N1$
590 PRINT"1. Street Address: ";A1$
600 PRINT"2. City/State: ";C1$:PRINT"3. Video Width: ";CVI(V1$):
PRINT"4.Line Feeds: ";L1$:PRINT"5. PASSWORD: ";P1$:PRINT"6. ACCE
SS: ";A2$
610 PRINT"Number of calls: ";CVI(C2$):PRINT"Number of messages:"
;CVI(M1$)
620 PRINT:PRINT"Tap ENTER to record":PRINT"Type a number to alte
r";
630 GOSUB130:PRINTCHR$(17):IF CT=0 THEN CLOSE:RETURN
640 IF VAL(I$)>6 OR VAL(I$)<1 THEN 620
650 ONVAL(I$)GOSUB670,690,710,730,760,780
660 PUT 2,MR:GOTO570
670 TL=0:PRINT"Address: ";:GOSUB130:TL=-1:PRINTCHR$(17):IF CT>0
THENLSETA1$=I$
680 RETURN
690 TL=0:PRINT"City/State: ";:GOSUB130:TL=-1:PRINTCHR$(17):IF CT
>0 THENLSETC1$=I$
700 RETURN
710 PRINT"Video Width: ";:GOSUB130:PRINTCHR$(17):IF VAL(I$)>20 A
NDVAL(I$)<255 THEN LSET V1$=MKI$(VAL(I$)):IF M=MR THEN SV=VAL(I$
):POKEVW,SV
720 RETURN
730 IF L1$="Y" THEN LSET L1$="N" ELSE LSET L1$="Y"
740 IF MR=M THENUL$=L1$:IF L1$="Y" THEN POKE PF,255 ELSE POKE PF
,0
750 RETURN
760 PRINT"PASSWORD: ";:GOSUB130:PRINTCHR$(17):IF CT>0 LSETP1$=I$
770 RETURN
780 IF SY THEN PRINT"Access: ";:GOSUB130:PRINTCHR$(17):LSET A2$=
I$
790 RETURN
800 PRINT"Section (Tap ENTER for all, ? forlist)";:GOSUB130:PRIN
TCHR$(17);
810 IF CT=0 THEN SE$=UA$:RETURN
820 IPASC(I$)=63THENGOSUB830:GOTO800
821SE$=LEFT$(I$,CT):FORZ=1TOCT:IFINSTR(UA$,MID$(SE$,Z,1))=0THENP
RINT"Unauthrized to Section ";MID$(SE$,Z,1):GOTO800ELSENEXT:RETU
RN
830 FOR Z=1 TO 15:IF (S$(Z)=STRING$(16,32)) OR(INSTR(UA$,MID$("0
12345678 9ABCDE",Z,1))=0) THEN 850
840 PRINT MID$("012345678 9ABCDE",Z,1);" ";S$(Z)
850 NEXT:RETURN
860 N$=FN FL$(MN):MID$(A$,INSTR(A$,"/")-LEN(N$),LEN(N$))=N$:RETU
RN
870 Y=1:IFEN=1 THEN RETURN
880 GET 2,Y
890 IF CK$>N1$ THEN FS=CVI(RP$):IF FS=0 THEN LSET RP$=MKI$(EN-Y)
```

Listing continued

Listing continued

```
:PUT2,Y:RETURN
900 IF CK$<=N1$ THEN FS=CVI(LP$):IF FS=0 THEN LSET LP$=MKI$(EN-Y
):PUT2,Y:RETURN
910 Y=Y+FS:GOTO880
920 LSET N1$=N5$:LSET A1$=AD$:LSET C1$=CT$:LSET L1$=CL$:LSETA2$=
CA$:LSET M1$=MKI$(0):LSET C2$=MKI$(1):LSET P1$=PW$:LSETUH$=MKI$(
0):LSET V1$=MKI$(CV):LSET LP$=MKI$(0):LSET RP$=MKI$(0)
930 PUT 2,NM:RETURN
1090 N5$="":TL=-1:PRINT"Name: ";:GOSUB130:PRINTCHR$(17):IF CT=0
THENRETURN ELSE N5$=LEFT$(I$,CT)
1100 LSET CK$=N5$:EN=NM:GOSUB940:IF NOT ER THEN PRINT"Name inuse
.":PRINT"Please add an initial.":GOTO1090
1110 TL=0:PRINT"Street Address: ";:GOSUB130:PRINTCHR$(17):IF CT>
0 THENAD$=LEFT$(I$,CT) ELSE 1110
1120 PRINT"City/State/Zip: ";:GOSUB130:PRINTCHR$(17):IF CT>0 THE
NCT$=LEFT$(I$,CT) ELSE 1120
1130 CLS:PRINTCHR$(12);N5$:PRINTAD$:PRINTCT$:PRINT:PRINT"Is this
 correct(y/n)?";
1140 TL=-1:GOSUB130:PRINTCHR$(17)
1150 IF ASC(I$)=78 THEN 1090 '    "NO"
1160 IF ASC(I$)<>89 THEN 1140 '  INCORRECT RESPONSE
1170 RETURN
1235T!=VAL(MID$(TM$,10))*3600+VAL(MID$(TM$,13))*60+VAL(MID$(TM$,
16)):RETURN
1240 ONERRORGOTO5190
1250 FORX=1TO20:MG$(X)=STRING$(80,32):NEXT
1260 GOSUB190
1270 GET 3,1
1280SL=CVI(SL$):SH=CVI(SH$):CA$=SA$:CV=CVI(SV$):SN=CVI(SN$):PW$=
SP$:CL$=SP$:X=CVI(SM$):SC=CVI(SC$):ND=CVI(ND$):NM=CVI(NM$):DS=CV
I(DS$)
1290 POKE VW,SV:IF UL$="N" THEN POKE PF,0
1300 GOSUB210
1310 GET3,2
1320 FOR X=1TO16:S$(X)=F$(X):NEXT
1330 FIELD 3,255 AS ZM$:GET 3,3:MN$=LEFT$(ZM$,2*SN):CLOSE
1340 CLS:FOR X=0 TO 127:SET(X,0):SET(X,47):IF X<48 THENSET(0,X):
SET(127,X):NEXT ELSE NEXT:PRINT@130,TIME$;
1350 PRINT@334,"The Saturday Software BulletinBoard";:PRINT@410,
"Version 2.0";:PRINT@452,"Copyright (c) 1984 by J.Stewart Schnei
der and C. E. Bowen";:PRINT@178,"Chat Mode";:IFCHTHENPRINT"ON ";
ELSEPRINT"OFF";
1355 PRINT@587,"<S>et Date/Time                <C>hat Toggle";:PR
INT@651,"<L>ocal Operation              <E>xit Towne Crier";
1360 IF (INP(&HE8)AND32)=0 THEN 1380
1365 A$=INKEY$:N=PEEK(&H387F):IFN=0THEN1360
1375 IFA$<>""THENA$=CHR$(ASC(A$)AND95):GOTO6500 ELSE1360
1380 CLS:PRINT:PRINTCHR$(12);"Hello, and welcome to":PRINT"the T
he TowneCrier BBS! (Ver. 2.0)":PRINT:PRINT
1390 FOR X=1 TO 5
1400 IFINKEY$<>""THEN1400
1410 NEXT:IN$=TIME$:TM$=IN$:GOSUB 1235:IN1=T!
1420 CLOSE:PRINT"Are you a first-time visitor?(y/n)";:GOSUB130:P
RINTCHR$(17)
1430 IF ASC(I$)=89 THEN 1530 ' YES
1440 IF ASC(I$)<>78 THEN 1420
1450 PRINT"Name: ";:GOSUB130:PRINTCHR$(17):IF CT=0 THEN 1420 ELS
ENA$=LEFT$(I$,CT)
1460 PRINT"Checking Membership Log...":CLOSE:GOSUB240:LSETCK$=NA
$:EN=NM:GOSUB940
1470 IF ER THEN PRINT"Not recorded":GOTO1420
1480 IF PW$=P1$ THEN 1515
1490 FOR TR=1 TO 3:PRINT"PASSWORD: ";
1500POKENP-1,0:POKENP,0:GOSUB130:PRINTCHR$(17):POKENP-1,&HCD:POK
ENP,&H33
1510 IF LEFT$(I$,16)<>P1$ THEN 1520
1515M=MR:AD$=A1$:CT$=C1$:SV=CVI(V1$):M2=CVI(M1$):UA$=A2$:HI=CVI(
UH$):HM=HI:I L1$="N"THEN POKE PF,0 ELSE POKE PF,255
1516 GOTO1630
1520 NEXT:PRINT"INVALID LOGIN":GOTO1420
1530 IF LEFT$(PW$,8)="PASSWORD" THEN 1600
1540 FOR TR=1 TO 3
1550 PRINT"System password? ";:GOSUB130:PRINTCHR$(17)
1560 IF LEFT$(I$,16)=PW$ THEN 1600
1570 NEXT:PRINT"Invalid login"
1580 PRINT"Board closed to unauthorized persons.":PRINT"Please h
ang up."
1590 GOSUB130:GOTO1590
1600 GOSUB240:GOSUB1090:IF N5$="" THEN 1420 ELSE NA$=N5$
1610 SV=CV:POKE VW,SV:UA$=CA$:NM=NM+1:GOSUB920:M=NM:IF M>1 THENE
N=M:GOSUB870
1620 MR=M:GOSUB570:CLOSE
1630 IF (WZ) OR (INSTR(UA$,"*")>0) THEN UA$="0123456789ABCDE*":S
Y=-1
1640 SE$=UA$:Q$="ALL"+CHR$(128):PRINT"One momentplease...":PRINT
```

Listing continued

BBS EXPRESS

Listing continued

```
"Checking your mailbox.":GOSUB220:SF=-1:MN=HI+1:GOSUB430
1650 FOR RN=S TO SN:PRINT".";:GET1,RN:L=20:IF (ASC(S2$) AND 16)=
16 THEN1670
1656 IF (L>0) AND (MID$(T1$,L,1)=" ")THEN L=L-1:GOTO 1656
1660 IF (SY AND (LEFT$(T1$,5)="SYSOP")) OR (LEFT$(T1$,L)=NA$) OR
(LEFT$(T1$,LEN(Q$))=Q$) THENGOSUB350:IFRD THEN PRINT:PM=PM+1:M(P
M)=RN:GOSUB4720
1670 NEXT:CLOSE:SF=0:IF PM>0 THEN PRINT"These message(s)":PRINT"
havebeen marked for retrieval":PRINT"Using the RM command.":PRIN
T
1680 PRINT"You are caller";SC:PRINT"SYSTEM CONTAINS";SN;"MESSAGE
S FROM"SL"TO"SH
1690 IF M>0 THEN PRINT"HIGH MESSAGE RETRIEVED: ";HI
1700 POKE VW,SV:IF UL$="N" THEN POKE PF,0
1710 F2$="BULLETIN":GOSUB4850
1720 SF=0:TL=-1:NC=0:CLOSE
1730 PRINT:PRINT"Command (? for HELP): ";:GOSUB130:PRINTCHR$(17)
:IF CT=0THEN 1730
1740 CM$=LEFT$(I$,CT):GOSUB300
1750ONINSTR("$RSLECXMUHBT",C$)GOTO1900,2865,2850,3280,2630,1790,
3710,1810,180,1780,1774,1777
1760 PRINT"? - Prints this list":PRINT"R - Read Messages":PRINT"
L -Leave a message":PRINT"S - Scan the messages":PRINT"E - Exit
theBBS":PRINT"C - Change Member's Record":PRINT"X - Database"
1770 PRINT"M - Scroll membership log":PRINT"U - Scroll user log"
:PRINT"H- View HELP file":PRINT"B - Set/Reset Brief Mode":PRINT"
    (No stopbetween messages)":PRINT"T - Talk to SYSOP":GOTO1720
1774 BM=(BM=0):PRINT"Brief mode ";:IF BM THEN PRINT"ON"ELSEPRINT
"OFF"
1776 GOTO1720
1777 IFNOTCHTHENPRINT"SYSOP not available":GOTO1720
1778 PRINT"PagingSYSOP.";:FORX=1TO10:PRINT".";:PORT=1TO50:IFPEEK
(&H387F)=0THENNEXTT,X:PRIT:PRINT"SYSOP doesn't answer.  Sorry":G
OTO1720ELSEPRINT"Control-P toexit CHAT"
1779 GOSUB130:PRINTCHR$(17);:IFINSTR(I$,CHR$(16))=0THEN1779ELSE1
720
1780 F2$="HELP":GOSUB4850:GOTO1720
1790 PRINT"Change Member's Record":GOSUB240
1800 MR=M:GOSUB570:GOTO1720
1810 GOSUB240:PRINT"Control-P to stop"
1820 FOR X=1 TO NM:GET 2,X
1830 PRINTN1$:PRINTC1$:PRINT
1840 IF INKEY$<>CHR$(16) THEN NEXT
1850 PRINT"Job done":CLOSE:GOTO1720
1860 GOSUB280:IF LOF(3)=0 THEN 1720 ELSEPRINT"Control-P to Stop"
:FORX=LOF(3) TO 1 STEP-1:GET 3,X:PRINTU1$:PRINT"IN: ";U2$:PRINT"
OUT:";U3$:PRINT"ELAPSED:    ";U4$:PRINT
1870 IF INKEY$=CHR$(16) THEN X=1
1880 NEXT
1890 GOTO1720
1900 IF NOT SY THEN 1720
1910 IFCM$<>""THEN 1930
1920 CLOSE:PRINT"SYSOP Commands (?=HELP): ";:GOSUB130:PRINTCHR$(
17):IFCT=0 THEN 1920 ELSE CM$=LEFT$(I$,CT)
1930 GOSUB300
1940ONINSTR("MACDELPSN$BFK",C$)GOTO2030,2100,2200,2130,1720,2240
,2570,2310,210,1980,1990,1975,1976
1950 PRINT"? - Prints this list":PRINT"A - Add New Member":PRINT
"C -Change User Record":PRINT"D - Delete User Record":PRINT"K -
Kill ERRORand USER logs"
1960 PRINT"E - Exit Sysop Function":PRINT"L - Print MailingLabel
s":PRINT"N - Set Section Names":PRINT"S - Examine/Change SystemD
efaults"
1970 PRINT"P - Purge Retrieved message":PRINT"M - Membershiplog"
:PRINT"B - Write Bulletin":PRINT"F - Rewrite Membership FilePoin
ters":GOTO1920
1975GOSUB240:E=NM:GOSUB5270:NM=NV:GOSUB190:GET3,1:LSETNM$=MKI$(N
M):PUT3,1:CLSE:GOTO1920
1976 KILL"ERROR/BBS"+SD$:OPEN"O",1,"ERROR/BBS"+SD$:PRINT#1,"ERRO
RLOG":CLOSE:KILL"USER/BBS"+SD$:GOTO1900
1980 STOP:GOTO 1720
1990 B=-1:GOSUB3380:B=0:OPEN"O",3,"BULLETIN/BBS"+DD$
2000 FOR Z=1TOLN
2010 IF MG(Z)>0 THEN PRINT#3,LEFT$(MG$(Z),MG(Z));
2020 NEXT Z:CLOSE 3:GOTO1920
2030 GOSUB1230
2040 GOSUB240
2050 FOR MR=1 TO NM:GET2,MR
2060 PRINTN1$:PRINTA1$:PRINTC1$:PRINT"PASSWORD: ";P1$:PRINT"Acce
ss:";A2$:PRINT"No. of Calls: ";CVI(C2$):PRINT"Number of Messages
:";CVI(M1$):PRINT
2070 IF PR THEN LPRINTN1$,"Password: ";P1$:LPRINTA1$,"Access:";A
2$:LPRINTC1$,"No. of Calls: "CVI(C2$):LPRINT"Video:";CVI(V1$),,"
Number of Messages: ";CVI(M1$):LPRINT"line feeds:";L1$:LPRINT" "
2080 IF INKEY$<>CHR$(16) THEN NEXT
```

Listing continued

Listing continued

```
2090 PRINT"Job done":CLOSE:GOTO1920
2100 PRINT"Add new member selected":GOSUB240:GOSUB1090:IF N5$="" THEN1900
2110 NM=NM+1:GOSUB920:MR=NM:IF MR>1 THEN EN=MR:GOSUB870
2120 GOSUB570:CLOSE:GOTO1920
2130 CLS:PRINTCHR$(12)"Delete User Record":GOSUB240
2140 PRINT"User Name: ";:GOSUB130:PRINTCHR$(17):IF CT=0 THEN 2190 ELSELSET CK$=I$
2150 EN=NM:GOSUB940:IF ER THEN PRINT"No such record":GOTO2140
2160 PRINT"Deleting ";N1$:PRINT"Confirm (y/n)? ";:GOSUB130:PRINTCHR$(17)
2170 IF ASC(I$)=89 THEN DF=-1:LSET P1$=CHR$(0):PUT 2,MR
2180 GOTO2140
2190 IF DF THEN E=NM:GOSUB5270:NM=NV:GOSUB190:GET 3,1:LSETNM$=MKI$(NM):PUT 3,1:DF=0
2195 GOTO1920
2200 CLS:PRINTCHR$(12)"Change User Record":GOSUB240
2210 PRINT"User Name: ";:GOSUB130:PRINTCHR$(17):IF CT=0 THEN 1920 ELSELSET CK$=I$
2220 EN=NM:GOSUB940:IF ER THEN PRINT"No such record":GOTO2210
2230 GOSUB570:GOTO1920
2240 CLS:PRINTCHR$(12);"Mailing Labels."
2250 GOSUB1230:IF NOT PR THEN PRINT"PRINTER NOT READY!!":GOTO1920
2260 GOSUB240
2270 FOR MR=1TONM:GET2,MR
2280 LPRINT N1$:LPRINT A1$:LPRINT C1$:LPRINT" ":LPRINT" ":LPRINT" "
2290 IF INKEY$<>CHR$(16) THEN NEXT
2300 PRINT"Job done":CLOSE:GOTO1920
2310 CLS:PRINTCHR$(12):GOSUB190
2320 GET 3,1:PRINT"1. Video width: ";CVI(SV$):PRINT"2. Line feeds:";SP$:PRINT"3. Default Access: ";SA$:PRINT"4. Maximum messages:";CVI(SM$):PRINT"5. System Password: ";SP$
2330 PRINT"6. Hard copy ";:IF PR THEN PRINT"On" ELSE PRINT"Off"
2340 PRINT"Tap ENTER to record":PRINT"TYPE A NUMBER TO ALTER"
2350 GOSUB130:PRINTCHR$(17):IF CT=0 THEN CLOSE:GOTO1910
2360 IF VAL(I$)<1 OR VAL(I$)>6 THEN 2320
2370 ON VAL(I$) GOTO2390,2410,2430,2450,2470,2490
2380 PUT 3,1:GOTO2320
2390 PRINT"VIDEO WIDTH: ";:GOSUB130:PRINTCHR$(17):IF (CT>0) AND(VAL(I$)>0) AND (VAL(I$)<81)THEN LSET SV$=MKI$(VAL(I$))
2400 GOTO2380
2410 IF PEEK(PP)=255 THEN POKE PP,0:LSET SP$="N" ELSE POKE PP,255:LSETSP$="Y"
2420 GOTO2380
2430 PRINT"SYSTEM ACCESS: ";:GOSUB130:PRINTCHR$(17):IF CT>0THEN LSETSA$=I$
2440 GOTO2380
2450 PRINT"MAXIMUM MESSAGES: ";:GOSUB130:PRINTCHR$(17):IF (CT>0) AND(VAL(I$)>10) AND (VAL(I$)<100) THEN LSET SM$=MKI$(VAL(I$)):MX=VAL(I$)
2460 GOTO2380
2470 PRINT"System password: ";:GOSUB130:PRINTCHR$(17):IF CT>0 THEN LSETSP$=I$
2480 GOTO2380
2490 IF PR THEN PR=0 ELSE PR=-1
2500 GOTO2380
2510 PRINT"Tap ENTER to leave unchanged":PRINT"or type new header.":TL=0
2520 FOR X=1 TO 15:PRINTMID$("0123456789ABCDE",X,1);" ";S$(X)
2530 GOSUB130:PRINTCHR$(17):IF CT=0 THEN 2560
2540 IF ASC(I$)=16 THEN X=15:GOTO2560
2550 LSET S$(X)=I$
2560 NEXT:GOSUB190:GOSUB210:FORX=1 TO 15:LSET F$(X)=S$(X):NEXT:PUT3,2:CLOSE:TL=-1:GOTO1920
2570 GOSUB220
2580 PRINT"Ready to PURGE recovered messages.":PRINT"Confirm(y/n)?";:GOSUB130:PRINTCHR$(17):IF ASC(I$)<>89 THEN 1900
2590 PRINT"WORKING";:FOR RN=1 TO SN:GET 1,RN
2600 IF (ASC(S2$) AND 16) =16 THEN MID$(MN$,2*RN-1,2)=MKI$(-PNP(RN,MN$))
2610 PRINT".";
2620 NEXT:PRINT:CLOSE:GOTO1920
2630 CLOSE:PRINTCHR$(17);"Thanks for calling The Towne Crier.";PRINT"Docall again.":PRINT"Arrival: ";IN$:OT$=TIME$:PRINT"Leaving:";OT$;TM$=OT$:GOSUB 1235
2640 E1=T1-IN1:Z=INT(E1/3600):E1=E1-Z*3600:Y=INT(E1/60):X=CINT(E1-Y*60)
2650 A$=FN FL$(Z):B$=FN FL$(Y):C$=FN FL$(X)
2660 IF LEN(A$)<2 THEN A$="0"+A$:GOTO2660
2670 IF LEN(B$)<2 THEN B$="0"+B$:GOTO2670
2680 IF LEN(C$)<2 THEN C$="0"+C$:GOTO2680
2690 PRINT"Elapsed time        ";A$;":";B$;":";C$
2700 PRINT"Please hang up now."
2710 IF M>0 THEN GOSUB240:GET2,M:LSET UH$=MKI$(HI):LSETC2$=MKI$(
```

Listing continued

Listing continued

```
CVI(C2$)+1):LSET M1$=MKI$(M2):PUT 2,M:CLOSE
2720 GOSUB280:LSET U1$=NA$:LSET U2$=IN$:LSET U3$=OT$:LSETU4$=A$+
";"+B$+";"+C$:PUT 3,LOF(3)+1:CLOSE
2850 CLS:PRINTCHR$(12);"SCANNING MESSAGE BOARD"
2860 PRINT"Enter a control-C to stop.":SF=-1
2865 IF SN=0 THEN PRINT"No messages available":GOTO 1720
2870 IPCM$<>""THEN2890
2880 PRINT"SUBCOMMAND (? for HELP): ";:GOSUB130:PRINTCHR$(17):IF
 CT=0THEN 1720 ELSE CM$=LEFT$(I$,CT)
2890 GOSUB300:D$=C$:IF D$="M" THEN 2970
2900 IF INSTR("FRISMN",D$)=0 THEN D$="?"
2910 IF D$="?" THEN PRINT"? - Prints this list":PRINT"F -Forward
":PRINT"R - Reverse"
2920 IF D$="?" THEN PRINT"I - Individual":PRINT"S - Selected":PR
INT"M -Marked":PRINT"N - New Messages":GOTO2880
2930 IF D$="N" THEN MN=HN+1:D$="F":GOTO2970
2940 MN=VAL(CM$):IF MN>0 THEN 2970
2950 PRINT"System contains messages"SL"to"SH:IF D$="I" THEN PRIN
T"Readwhich message?"; ELSE PRINT"Read starting with which messa
ge?"
2960 GOSUB130:PRINTCHR$(17):MN=VAL(I$)
2970 SE$=UA$:GOSUB220:IF D$<>"I" AND D$<>"M" THEN GOSUB800
2980 ON INSTR("FRISM",D$) GOTO2990,3040,3090,3130,3220
2990 E=SN:GOSUB430
3000 FOR RN=S TO E:GET 1,RN:GOSUB350:IF NOT RD THEN PRINT".";:GO
TO3020
3010 PRINT:GOSUB4720:GOSUB440:IF N>HI THEN HI=N
3020 IF INKEY$=CHR$(3) THEN RN=E
3030 NEXT:GOTO1720
3040 E=1:GOSUB430
3050 FOR RN=S TO E STEP-1:GET 1,RN:GOSUB350:IF NOT RD THENPRINT"
.";:GOTO3070
3060 PRINT:GOSUB4720:GOSUB440:IF N>HI THEN HI=N
3070 IF INKEY$=CHR$(3) THEN RN=E
3080 NEXT:GOTO1720
3090 GOSUB430:IF FN P(S,MN$)<>MN THEN PRINT"NO SUCH MESSAGE":GOT
O1720
3100 RN=S:GET 1,RN:GOSUB350
3110 IF RD THEN GOSUB4720:GOSUB440:IF N>HI THEN HI=N
3120 GOTO1720
3130 PRINT"Search field:(T,F,S)";:GOSUB130:PRINTCHR$(17):IF CT=0
 THEN1720 ELSE C$="P"
3140 SF$=LEFT$(I$,1):IF INSTR("TFS",SF$)=0 THEN 3130
3150 PRINT"Search for:";:GOSUB130:PRINTCHR$(17):SS$=LEFT$(I$,CT)
:PRINT"Searching";
3160 E=SN:GOSUB430
3170 FOR RN=S TO E:GET 1,RN:GOSUB350:IF NOT RD THEN PRINT".";:GO
TO3200
3180 PRINT:GOSUB4720:GOSUB440:IF N>HI THEN HI=N
3190 PRINT".";
3200 IF INKEY$=CHR$(3) THEN RN=E
3210 NEXT:SF$="":SS$="":GOTO1720
3220 E=PM:IF E=0 THEN PRINT"No Marked Messages":GOTO1720
3230 FOR X=1 TO E:RN=M(X):GET 1,RN:GOSUB350
3240 GOSUB4720:GOSUB440:IF N>HI THEN HI=N
3250 IF LEFT$(I$,1)="T" THEN X=E
3260 IF INKEY$=CHR$(3) THEN X=E
3270 NEXT:GOTO1720
3305 IF (TT$="ALL") AND (SY) THEN TT$=TT$+CHR$(128)
5190 IF ERL=4450 THEN PRINT"File not found":CLOSE:RESUME 4430
5200 IF ERL=4450 THEN PRINT"No such file":CLOSE:RESUME 4430
```

End

Uploading and Downloading The Express Way

Any data base is a constant source of stress on your software, especially when it's part of the software that keeps your bulletin board system (BBS) up and running. The most demanding part of any BBS, the data base, is where callers use bulk transfers to upload and download files. Now that your board is so close to completion, it's time that we discussed the data base in detail, with code included.

To achieve a better understanding of the data base's involvement, first consider data storage and transmission. Your BBS stores bytes received from callers in a Basic string before writing them to disk. As you saw in earlier issues of the BBS Express, your board responds to an XOFF character in time to stop data transmission so that characters in transit to or from the caller aren't lost.

This is important when you consider a 300-baud transmission. At 300 baud, there's a bit coming in approximately every .0033 seconds. Electricity travels about 621 miles in this time span; if the call is routed through a satellite whose round trip orbit is 45,000 miles, it takes an electrical signal about a quarter second to make the trip. This translates to the time it takes to transmit seven characters, meaning that the BBS has to reserve at least that amount of space to catch the pipeline material.

In terms of the data base, this is important because material transmits at computer speed, not typing speed. Before the BBS receives the material, however, it needs a file name to assure a place on the board's data base cata-

Illustration by Charlie Wilton

Program Listing. BBS module for the data base catalog.

```
1180 GOSUB260
1190 AC=0:LSET CK$=CM$:PRINT"Searching...":EN=DS:GOSUB940:IF ER
THEN CLOSE:PRINT"No such file.":RETURN
1200 IF (WZ) OR (P1$="PASSWORD        ") THEN 1220
1210 PRINT"FILE PASSWORD PROTECTED":PRINT"Password: ";:GOSUB130:
PRINTCHR$(17):IF LEFT$(I$,16)<>P1$ THEN PRINT"Access denied":RET
URN
1220 AC=-1:RETURN
3710 IF CM$<>"" THEN 3730
3720 CLOSE:PRINT"ACCESS Command (? for HELP): ";:GOSUB130:PRINTC
HR$(17):IF CT=0 THEN 3720 ELSE CM$=LEFT$(I$,CT)
3730 GOSUB300
3740 ONINSTR("$UDEC",C$)GOTO4160,3760,3960,1720,4020
3750 PRINT"? - Prints this list":PRINT"U - Upload to Towne Crier
":PRINT"D - Download from Towne Crier":PRINT"E - Exit XA space":
PRINT"C - Catalog":GOTO3720
3760 CC!=0:GOSUB260
3770 IF CM$<>"" THEN 3790
3780 PRINT"Filename:(20 max)";:GOSUB130:PRINTCHR$(17):IF CT=0 TH
EN 3720 ELSE CM$=LEFT$(I$,CT)
3790 LSET CK$=CM$:EN=DS:GOSUB940:IF ER THEN 3810
3800 PRINT"That name is in use.":PRINT"Please choose another.":G
OTO3780
3810 TL=0:PRINT"Description:(128 max)";:GOSUB130:PRINTCHR$(17):T
L=-1:IF CT=0 THEN 3810 ELSE DC$=LEFT$(I$,CT)
3820 CLS:PRINTCHR$(12);CK$:PRINTDC$:PRINT"Correct (y/n)? ";
3830 GOSUB130:PRINTCHR$(17)
3840 IF ASC(I$)=78 THEN 3780
3850 IF ASC(I$)<>89 THEN 3830
3860 A$="XA000/BBS"+DD$:ND=ND+1:MN=ND:GOSUB860:LSET F2$=A$:LSET
N1$=CK$:LSET D1$=DC$:LSET XA$=MKI$(0):LSET P1$="PASSWORD"
3870 IF SY THEN TL=-1:PRINT"Password (Default to none)";:GOSUB13
0:PRINT CHR$(17):IF CT>0 THEN LSET P1$=I$
3880 DS=DS+1:PUT 2,DS
3890 CLS:PRINTCHR$(12);"One moment please...":TL=0:NC=-1
3900 OPEN"O",3,A$
3910 PRINT"Ready to receive.":PRINT"Enter /EX to end."
3920 GOSUB130
3930 IFLEFT$(I$,3)="/EX"ORLEFT$(I$,3)="/ex"THEN3950
3940 PRINT#3,LEFT$(I$,CT);:CC!=CC!+CT:GOTO3920
3950 GET 2,DS:LSET LP$=MKI$(0):LSET RP$=MKI$(0):LSET XL$=MK$$(CC
!):PUT 2,DS:TL=-1:NC=0:PRINTCHR$(17);:EN=DS:GOSUB870:CLOSE:GOTO3
720
3960 IF DS=0 THEN PRINT"No data files available.":GOTO3720
3970 IF CM$<>"" THEN 4000
3980 PRINT"Filename to download ";:GOSUB130:PRINTCHR$(17):IF CT=
0 THEN 3720 ELSE CM$=LEFT$(I$,CT)
```

Listing continued

System Requirements

Model III (TRS-80 LOAD 80)
48K RAM
Disk Basic
Two disk drives

Listing continued

```
4000 GOSUB1180:IF NOTAC THEN 3720
4010 LSET XA$=MKI$(CVI(XA$)+1):PUT 2,MR:OPEN"I",3,F2$+DD$:GOSUB4
860:GOTO3720
4020 GOSUB260:GOSUB4030:GOTO3720
4030 IF DS>0 THEN CLS:PRINTCHR$(12);"FILE CATALOG" ELSE PRINT"No
 data files available":RETURN
4040 PRINT"1 - Catalog and description":PRINT"2 - No description
":PRINT"3 - Instring catalog"
4050 PRINT"Choose by number ";
4060 CA=VAL(CH$):IF CA=1 OR CA=2 OR CA=3THEN PRINT CA:GOTO4080
4070 GOSUB130:PRINT CHR$(17):IF CT=0 THEN RETURN ELSE CA=VAL(I$)
:IF CA<>1 AND CA<>2 AND CA<>3THEN 4050
4080 PRINT"Enter a Control-P to quit":IF (CA=2) OR (CA=3) THEN P
RINT"Filename                Size     Accesses"
4085 IF CA=3THENPRINT"Search for: ";:GOSUB130:PRINTCHR$(17);:SF$
=LEFT$(I$,CT)
4090 GOSUB1020
4100 IF ER THEN RETURN
4110 IF (NOTSY) AND (LEFT$(P1$,8)<>"PASSWORD") THEN 4140
4120 IF (CA=2) OR ((CA=3) AND (INSTR(N1$,SF$)>0))THEN PRINTUSING
U$;N1$;CVS(XL$);CVI(XA$);:IFNZTHENPRINT"  ";F2$ELSEPRINT
4130 IF CA=1 THEN PRINT N1$:POKE RE,SV:Z9=USR2(VARPTR(D1$)):PRIN
T D1$
4140 IF INKEY$=CHR$(16) THEN RETURN
4150 GOSUB1080:GOTO4100
4160 IF NOT SY THEN 3720
4170 CLOSE:PRINT"SYSOP Commands (?=HELP): ";:GOSUB130:PRINTCHR$(
17):IF CT=0 THEN 3720 ELSE CM$=LEFT$(I$,CT)
4180 GOSUB300
4190 ON INSTR("KHERCSPF",C$) GOTO4220,4310,3720,4370,4210,4430,4
660,4205
4200 PRINT"? - Prints this list":PRINT"K - KILL file":PRINT"H -
Edit header":PRINT"E - Exit SYSOP function":PRINT"R - Rename fil
e"
4202 PRINT"S - Submit file from disk":PRINT"P - Change Password"
:PRINT"C - Catalog":PRINT"F - Rewrite File Pointers":GOTO4170
4205 GOSUB260:E=DS:GOSUB5270:CLOSE:DS=NV:GOSUB190:GET3,1:LSETDS$
=MKI$(DS):PUT3,1:GOTO3710
4210 GOSUB260:GOSUB4030:GOTO4170
4220 GOSUB260:IFCM$<>""THEN4240
4230 PRINT"Filename to KILL (or ENTER to quit): ";:GOSUB130:PRIN
TCHR$(17):IF CT=0 THEN 4290 ELSE CM$=LEFT$(I$,CT)
4240 FL$=CM$:GOSUB1190:IF NOTAC THEN 4230
4250 PRINT"KILLING ";N1$:PRINT"PLEASE CONFIRM CONTINUATION (Y/N)
 ";
4260 GOSUB130:PRINTCHR$(17)
4270 IF ASC(I$)=89 THEN DF=-1:LSET P1$=CHR$(0):PUT 2,MR:CLOSE:KI
LL F2$:GOSUB260
4280 GOTO4230
4290 IF DF THEN E=DS:GOSUB5270:DS=NV:GOSUB190:GET3,1:LSET DS$=MK
I$(DS):LSETND$=MKI$(ND):PUT3,1:DF=0
4300 GOTO4170
4310 IF CM$<>""THEN4330
4320 PRINT"Filename to Edit: ";:GOSUB130:PRINTCHR$(17):IF CT=0 T
HEN 4170 ELSE CM$=LEFT$(I$,CT)
4330 GOSUB1180:IF NOTAC THEN 4170
4340 PRINT"Current description: ":PRINT D1$
4350 TL=0:PRINT"New description (128 max) ";:GOSUB130:PRINT CHR$
(17):TL=-1:IF CT>0 THEN LSET D1$=I$:PUT 2,MR
4360 GOTO4170
4370 IFCM$<>"" THEN 4390
4380 PRINT"Filename to RENAME: ";:GOSUB130:PRINTCHR$(17):IF CT=0
 THEN 4170 ELSE CM$=LEFT$(I$,CT)
4390 GOSUB1180:IF AC THEN X=MR ELSE 4170
4400 PRINT"New file name: ";:GOSUB130:PRINTCHR$(17):IF CT=0 THEN
 4170
4410 LSET CK$=I$:EN=DS:GOSUB940:IF NOTER THEN PRINT"Name in use.
":GOTO4400
4420 GET2,X:P$=P1$:LSET P1$=CHR$(0):PUT 2,X:LSET N1$=I$:LSET P1$
=P$:LSET LP$=MKI$(0):LSET RP$=MKI$(0):DS=DS+1:PUT 2,DS:DF=-1:GOT
O4290
4430 PRINT"Name of existing file to be submitted: ";:GOSUB130:PR
INTCHR$(17):IF CT=0 THEN 4170 ELSE FL$=LEFT$(I$,CT)
4440 GOSUB260
4450 OPEN"I",1,FL$:CLOSE1
4460 PRINT"Name of XA file: ";:GOSUB130:PRINTCHR$(17):IF CT=0 TH
EN 4170
4470 LSET CK$=I$:EN=DS:GOSUB940:IF NOTER THEN CLOSE:PRINT"Name i
n use.":GOTO4460
4480 TL=0:PRINT"Description:(128 max) ";:GOSUB130:PRINTCHR$(17):
IF  CT=0 THEN 4170 ELSE DC$=LEFT$(I$,128)
4490 A$="XA000/BBS"+DD$:ND=ND+1:NN=ND:GOSUB860:LSET F2$=A$:LSETN
1$=CK$:LSET D1$=DC$:LSET XA$=MKI$(0):LSETP1$="PASSWORD":LSET LP$
=MKI$(0):LSET RP$=MKI$(0)
4500 TL=-1:PRINT"PASSWORD (DEFAULT TO NONE)";:GOSUB130:PRINTCHR$
(17):IF CT>0 THEN LSET P1$=I$
4510 DS=DS+1:PUT 2,DS:PRINT"One moment please...":CC!=0:TL=0:NC=
-1
4520 OPEN"R",1,FL$:CC!=0
4530 FIELD 1,255 AS A1$,1 AS B1$
4540 OPEN"O",3,A$
4550 FOR X=1 TO LOF(1)
4560 GET 1,X
4570 PRINTA1$;:A2$=A1$:B$=B1$
4580 CR=INSTR(A2$,CHR$(13)):IFCR>0THENMID$(A2$,CR,1)=CHR$(141):G
OTO4580
4590 IFASC(B$)=13THENB$=CHR$(141)
4600 REM
4610 CR=INSTR(A2$,CHR$(0)):IF CR>0 THENCC!=CC!+CR:PRINT#3,LEFT$(
A2$,CR);:GOTO4640ELSEPRINT#3,A2$;:CC!=CC!+255
4620 IFB$<>CHR$(0)THENPRINT#3,B$;:CC!=CC!+1
4630 REM
4640 NEXT:PRINT
4650 CLOSE1:GET 2,DS:LSET XL$=MKS$(CC!):PUT 2,DS:TL=-1:NC=0:PRIN
TCHR$(17);:EN=DS:GOSUB870:GOTO4170
```

Listing continued

log. Messages, as discussed in some detail earlier, take on secret file names. The idea is to never give a caller access to the real file names on the disk; otherwise, your system is much too vulnerable.

Each data base file, then, has a public file name and a secret file name. The BBS builds the secret file name from the data base file number, XAxx/BBS, in the same way that it builds the message file. The first file is XA001/BBS, the second XA002/BBS, and so on. One advantage of this method is that your public file names aren't limited to legal TRSDOS file names; you're allowed 20 characters, all of which are legal.

Binary Trees Revisited

In creating the data base catalog, you'll also use the binary tree method that was used to find files in the membership log (February 1985, p. 104). You must make each file name unique, though, because duplicate file names might confuse the tree. When a caller orders an upload, he's first prompted for a file name, which is then evaluated for originality.

Lines 3760–3850 of the Program Listing perform this operation. In addition, since you're maintaining your own directory, line 3810 makes it possible to include information such as descriptions, which aren't normally included in a DOS directory.

Line 3860 builds the secret file name by calling the same routine that creates the secret names for the message board. If SY = −1, line 3870 gives you the option of password protecting the file you're about to upload. If the file's password is anything but "Password," it's not visible on the catalog.

Line 3880 bumps the data slot (DS) by 1, and puts the information for this file in XASPACE/BBS. DS is the number of the last record used in XASPACE/BBS. Line 3920 opens the secret file, A$, while line 3920 calls the telecommunicating input statement. Each call stores up to 255 characters in I$, and then exits after sending an XOFF. If the line begins with /EX (or /ex), the BBS terminates the routine in line 3930. Otherwise, you'll use PRINT#3 to write the string to the secret file, and bump CC! by the value of CT, the character count.

When the caller signals his exit with

Listing continued

```
4660 IFCM$<>""THEN4680
4670 PRINT"Filename: ";:GOSUB1130:PRINT CHR$(17):IF CT=0 THEN 417
0 ELSE CN$=LEFT$(I$,CT)
4680 GOSUB1180:IF NOTAC THEN 4170
4690 LSETP1$="PASSWORD":PRINT"New Password:(Default to none)  ";
:GOSUB130:PRINT CHR$(17):IF CT>0 THEN LSET P1$=I$
4700 PUT2,MR
4710 GOTO4170
5190 IF ERL=4450 THEN PRINT"File not found":CLOSE:RESUME 4430
5210 CLOSE:PRINT"%% System error in ";ERL:PRINT"Logging error.
One moment please..."
5260 CLOSE:RESUME 1720
5270 NV=0:PRINT"File Compression":PRINTE"Files"
5280 FOR X=1 TO E:GET2,X:PRINTX;N1$;
5290 IF ASC(P1$)=0 THEN PRINT"-->DELETED";:GOTO5320
5300 NV=NV+1:LSET LP$=MKI$(0):LSET RP$=MKI$(0):PUT2,NV
5310 IF NV>1 THEN LSETCK$=N1$:EN=NV:GOSUB870
5320 PRINT:NEXT
5330 IF NV=LOF(2)THEN5350
5340 FOR X=NV+1 TO LOF(2):GET2,X:LSETLP$=MKI$(0):LSETRP$=MKI$(0)
:PUT2,X:NEXT
5350 PRINT"Compressed to"NV"Files":CLOSE:RETURN
6500 ONINSTR("SCLE",A$) GOTO 6520,6570,6580,6590
6510 GOTO 1350
6520 CLS:LINEINPUT"Enter TIME$ desired (MM/DD/YY HH:MM:SS) ";TM$
6530 IFVAL(TM$)<1ORVAL(TM$)>12ORVAL(MID$(TM$,4))<1 ORVAL(MID$(TM
$,4))>31ORVAL(MID$(TM$,10))<0 ORVAL(MID$(TM$,10))>23ORVAL(MID$(T
M$,13))<0ORVAL(MID$(TM$,13))>59ORVAL(MID$(TM$,16))<0 OR VAL(MID$
(TM$,16))>59 THEN 6520
6540 IF MID$(TM$,3,1)<>"/" OR MID$(TM$,6,1)<>"/" OR MID$(TM$,9,1
)<>" " OR MID$(TM$,12,1)<>":" OR MID$(TM$,15,1)<>":" THEN 6520
6550 Y=0:FORX=1TO16STEP3:POKE16924-Y,VAL(MID$(TM$,X,2)):Y=Y+1:NE
XT:CLS:GOTO1340
6570 CH=(CH=0):IFCHTHENPRINT@188,"ON ";:POKE-1,1ELSEPRINT@188,"O
FF";:POKE-1,0
6575 GOTO1360
6580 POKETT,0:POKETT+1,0:POKETN,0:POKETN+1,0:WZ=-1:SY=-1:GOTO138
0
6590 CLOSE:END
```

End

/EX, line 3950 gets the record from XASPACE/BBS, and LSETS LP$ and RP$, the binary tree pointers, to zero. It also LSETS XL$, the length of the file, to CC! and puts the record back. The GOSUB 870 command adds the new file to the existing binary tree. The PRINT CHR$(17) commands scattered about simply send XON characters to the caller.

Ready to Download

Downloading files from the data base is a much simpler proposition. When downloading, the caller's terminal program must handle the timing, while the board watches for XOFF characters. Now, lines 1180–1220 decide whether or not to grant the caller access to the file. He must know the correct file name, and, if it's password-protected, he must know the password. If the caller is a wizard, that is, operating from the BBS's host console, he needn't know the password. This routine uses another Boolean variable, AC. If it's zero, the BBS doesn't grant access; if it's −1, the caller may see the file.

The downloading actually starts at line 3960 with a call to 1180. If the caller has access to the file, line 4010 bumps XA$, the number of accesses, to that file; it then opens the secret file name, F2$, and goes to line 4860, the same read routine used by the message base.

Now that you have all that information in XASPACE/BBS, you ought to be able to recover it. The catalog starts at line 4020 and calls a subroutine at line 4030. Because the catalog is a subroutine, the sysop can call it from the sysop menu without having to duplicate code. The BBS displays the catalog either with or without descriptions.

You also have the option to search through the catalog. If you do so, line 4085 initiates a search string. The BBS displays only those file names containing that specific string. Line 4110 makes the password-protected files invisible to non-sysops, and line 4120 prints the secret file name for wizards. By knowing the file name, a wizard can call the file into a word processor and edit it before saving it back to the same file name.

Calls to lines 1020 and 1080 initiate the routine that produces a sorted list, which results in an alphabetized list of the files. The call to USR2 in line 4130 video formats the description to the caller's video width, while line 4140 checks for a control-P from the caller.

Lines 4160 to 4710 handle these sysop options: kill, rename, and password-protect. Killing or renaming a file calls the file compression routine in lines 5270 to 5350, which works to maintain the integrity of the binary tree. Any file with a password of CHR$(0) is removed and the file is compressed to fill the hole left behind. ■

The BBS Express, 80 Micro*'s bulletin board system, is open 24 hours a day. Call us at 603-924-6985 to see the finished product. UART parameters are 300 baud, 7-bit words, one stop bit, and even parity.*

You can reach J. Stewart Schneider and Charles E. Bowen either through their bulletin board at 606-739-6088 or c/o Saturday Software, P.O. Box 404, Catlettsburg, KY 41129.

Log Off at 05:85 Connect Time: 13 Months

Installing and maintaining your own bulletin board system (BBS) isn't difficult, but it does require close attention to detail. Your attention should focus primarily on a telephone, a modem, a universal asynchronous receiver/transmitter (UART) a computer, and the BBS software. While each element has a unique function, smooth and efficient interaction among the components is of paramount importance.

Consider some potential trouble spots on the BBS. A poor telephone connection is the first thing to check if you're having problems. The telephone has to deliver a clear signal at all times; any static on the line will cause problems.

If the telephone is doing its job and you're still having problems, check out the modem. Simply put, there are good modems and bad modems; Radio Shack's Modem II falls into the second category. Though some people have been able to use the Modem II successfully, we've yet to find reliable solutions to the hang-up and reset problems characteristic of this modem. You can turn the modem off and then back on to initiate a reset, but this just isn't practical on a BBS. Try these commands if you own a Modem II.

```
OUT234,175:FOR X=1 TO 1000:NEXT:
  OUT 234,164
```

or

```
OUT234,180:FOR X=1 TO 1000:NEXT:
  OUT 234,164
```

If you're using the Hayes Smartmodem, set the front panel switches to UDUDUUU.

Some modems in half-duplex systems, like the Hayes, have a habit of echoing everything back to the computer. If the software echoes everything it receives to the modem and the modem echoes everything it receives to the software, you're stuck with an infinite loop. The cursor zips to the end of a 255-byte line and locks up your system. If this happens, turn off your modem's echo function.

Illustration by Charlie Wilton

Automatic Control

When you turn on your Model III, there's no connection between the communications line and the video or keyboard. This is the responsibility of Upload, the BBS's software linker. From TRSDOS Ready, type in UPLOAD. The TRSDOS Ready banner should return, though it'll run slowly. If, however, your cursor goes crazy, your modem is echoing.

Under LDOS, the procedure is a little different. You'll need this JCL file (LINKUP):

```
MEMORY (HIGH=X'FDFF')
UPLOAD
SET *KI KI (TYPE)
SET *CL RS232T (DTR=ON)
  (RS232R on the Model I)
LINK *KI *CL
LBASIC RUN "HOST/BAS"
//STOP
```

To run your BBS from LDOS, type in DO=LINKUP. LDOS is more convenient than TRSDOS here because you can apply an automatic boot-up command to the disk by typing in AUTO DO=LINKUP.

TRSDOS can't handle an automatic command because of memory conflicts. Instead, get into Basic, set the memory size at 65000, reserve three files, and type in RUN "HOST/BAS". Before entering Basic, however, you need to build a file. To do this, type in BUILD MSG0001/BBS:1 at the Ready prompt of either DOS. When the disk drive light goes out, type in the following: "Hi, everybody. Welcome to my BBS." Then press the enter key, followed by the break key.

Signing On

One of your first tasks after booting up the BBS is to log yourself on as the sysop. There's only one way to do this and that's from the console. Operating from the console is called the wizard mode. It gives you full access to

System Requirements

Model III LOAD 80
48K RAM
Disk Basic
Assembly language
Two disk drives
Editor/assembler

Program Listing 1. BBS machine-language assist module.

```
00050 ;Towne Crier machine code receive and assist module
00060 ;Copyright (c) 1984 Schneider and Bowen
00070 ;All rights reserved.  For 80-Micro Subscribers only
00080 ;No transfer, duplication or distribution rights
00090 ;granted.
00110 ;       RECEIVER SECTION 3/23/84
00120 ;       RECEIVES CHARACTERS FROM THE RS232 LINK
00130 ;       OR THE KEYBOARD, AND STORES THEM
00140 ;       SEQUENTIALLY IN I$ UNTIL THE LENGTH OF I$
00150 ;       IS MET, OR A CARRAIGE RETURN IS ENTERED.
00160 ;       RECOGNIZES BACKSPACE, AND RETURNS ERROR
CONDITION
00170 ;       ON LOSS OF USER CARRIER.
00180 ;
00190 ;
00200 VBLK    EQU     401DH           ;VIDEO DCB
00210 KBLK    EQU     4015H           ;KB DCB
00220 UDATA   EQU     0EBH            ;UART DATA PORT
00230 USTAT   EQU     0EAH            ;UART STATUS REGISTER
00240 MSTAT   EQU     0E8H            ;MODEM STATUS PORT
00250 PRINT   EQU     33H             ;ROM PRINT ROUTINE
00260 KEYBRD  EQU     2BH             ;ROM KEYBOARD SCAN
00270 BASIC   EQU     0A9AH           ;PASS TO BASIC
00280 TODOS   EQU     402DH           ;JUMP TO DOS
00290 TRSDOS  EQU     1               ;SET TO 0 FOR LDOS
00300 ;
00310 ;
00320         ORG     0FE00H
00330 RECV    CALL    PARAM           ;GET STRING VALUES
00340 MN      LD      B,245
00350         LD      C,0             ;ACCUMULATOR
00360         LD      A,11H           ;CONTROL-Q
00370         CALL    PRINT           ;SEND IT
00380 MAIN    IN      A,(MSTAT)       ;CHECK MODEM STATUS
00390         AND     32              ;CHECK CARRIER DETECT
00400 TN      JR      NZ,NOTONE       ;LOST THE TONE
00410         CALL    KEYBRD          ;CHECK EVERYTHING
ELSE
00420         OR      A
00430         JR      Z,MAIN          ;NOTHING
00440         CP      8               ;BKSPCE?
00450         JR      NZ,NOTBKS       ;NO
00460         LD      A,C
00470         OR      A               ;AT FIRST CHAR?
00480         JR      Z,MAIN          ;YES - IGNORE
00490         INC     B
00500         INC     B               ;FOR THE DECREMENT
00510         DEC     C
00520         DEC     C               ;FOR THE INCREMENT
00530         DEC     HL
00540         LD      (HL),20H
00550         DEC     HL
00560         LD      A,8
00570         JR      NP              ;PERFORM BKSPCE
00580 NOTBKS  LD      (HL),A          ;STORE CHARACTER
00590 NP      CALL    PRINT           ;PRINT IT
00600         INC     HL              ;FOR NEXT CHARACTER
00610         INC     C
00620         CP      0DH             ;TERMINATOR ENTERED?
00630         JR      Z,EXIT          ;TERMINATOR FOUND
00640 NOT010  DJNZ    MAIN            ;LOOP TIL DONE
00650 EXIT    LD      A,13H           ;CONTROL-S
00660         CALL    PRINT           ;SEND IT
00670         LD      B,255           ;DELAY
00680         LD      E,10            ;MAX # OF CHAR.
00690 EXI010  IN      A,(MSTAT)       ;CHECK MODEM STATUS
00700         AND     32              ;CARRIER DETECT
00710 TT      JR      NZ,NOTONE       ;LOST CARRIER
00720         CALL    KEYBRD          ;CHECK FOR INCOMING
00730         OR      A
00740         JR      Z,EXI030        ;NONE - EXIT
00750         INC     C               ;COUNT CHARACTER
00760         LD      (HL),A          ;STORE IT
00780         INC     HL              ;BUMP POINTER
00790         DEC     E               ;DECREMENT CHAR.
COUNT
00800         JR      Z,EXI020        ;MAX # RECEIVED
00810 EXI030  DJNZ    EXI010          ;DECREMENT TIMER
00820 EXI020  LD      L,C
00830         LD      H,0             ;TO PASS TO BASIC
00840         JP      BASIC
00850 NOTONE  LD      HL,-1
00860         JP      BASIC           ;PASS ERROR TO BASIC
00870 ;
00880 ;
00890 ;       CAPITALIZATION ROUTINE
00900 ;       CAPITALIZES CHARACTERS IN THE RANGE
00910 ;       96 < C < 123
00920 ;
00930         ORG     0FE6CH
00940 CAPIT   CALL    PARAM           ;GET VARPTR(I$)
00950 CAPL    LD      A,(HL)
00960         CP      97
00970         JR      C,NOCAP         ;C<97
00980         CP      123
00990         JR      NC,NOCAP        ;C>122
01000         AND     95              ;MAKE IT A CAPITAL
01010         LD      (HL),A          ;REPLACE IT
01020 NOCAP   INC     HL
01030         DJNZ    CAPL
01040         RET                     ;DONE
01050 ;
01060 ;       VIDEO SCANNER
01070 ;       RECEIVES STRING VARPTR AND VIDEO WIDTH
01080 ;       FROM BASIC, AND PARSES STRING TO CORRECT
01090 ;       WIDTH
01100 ;
01110 PARSE   CALL    PARAM           ;GET VARPTR(STRING)
01120         LD      A,(REMAIN)
01130         LD      C,A             ;REMAIN. CHAR. TO C
01140         LD      DE,0            ;CLEAR COUNTER
01150         LD      (SPACE),DE      ;CLEAR SPACE
INDICATOR
01160 SLOOP   LD      A,(HL)          ;GET CHAR.
01170         CP      20H             ;SPACE?
01180         JR      NZ,SL010        ;NO
01190         LD      (SPACE),HL
01200         LD      E,B             ;SAVE COUNTER &
ADDRESS
01210 SL010   CP      0DH             ;CAR. RET?
01220         JR      NZ,SL020        ;NOPE
01230         LD      DE,0
01240         LD      (SPACE),DE      ;CLEAR THINGS OUT
01250         LD      A,(VIDWIT)
01260         LD      C,A
01270         JR      ELOOP
01280 SL020   DEC     C
01290         JR      NZ,ELOOP        ;HAVEN'T REACHED END
01300         LD      A,(VIDWIT)
01310         LD      C,A             ;RESET VIDWIDTH
01320         LD      A,E
01330         OR      A               ;SPACE IN LINE?
01340         JR      Z,ELOOP         ;NOPE
01350         LD      HL,(SPACE)
01360         LD      (HL),0DH        ;INSERT C.R.
01370         LD      B,E             ;RESET COUNTER
01380         LD      DE,0
01390         LD      (SPACE),DE      ;RESET MARKER
01400 ELOOP   INC     HL              ;FOR NEXT CHAR.
01410         DJNZ    SLOOP           ;LOOP TIL DONE
01420         LD      A,C
01430         LD      (REMAIN),A      ;STORE REMAINING
01440         LD      A,E
01450         OR      A               ;SPACE IN LAST LINE?
01460         RET     Z               ;NOPE
01470         LD      A,C
01480         CP      5               ;REMAINING CHAR <=5?
01490         RET     NC              ;NOPE
01500         LD      A,(VIDWIT)
01510         ADD     A,E             ;A=VIDWIT+SPACE
COUNTER
01520         SUB     (IX+0)
;A=VIDWIT+SPACE-LEN(STR)
01530         LD      (REMAIN),A      ;RESET REMAINING
CHAR.
01540         LD      HL,(SPACE)
01550         LD      (HL),0DH        ;INSERT C.R.
01560         RET
01570 REMAIN  DEFB    0
01580 SPACE   DEFW    00
01590 VIDWIT  DEFB    0
01600 ;
01610 ;       PARAMETER RECOVERY SUBROUTINE
01620 ;       RECOVERS VARPTR(STRING) AND PLACES
01630 ;       LENGTH IN B, STRING ADDRESS IN HL
01640 ;       VARPTR(STRING) IN IX
01650 ;
01660 PARAM   CALL    0A7FH           ;GET VARPTR(STRING)
01670         PUSH    HL
01680         POP     IX              ;TO IX
01690         LD      B,(IX+0)        ;LENGTH TO B
01700         LD      L,(IX+1)
01710         LD      H,(IX+2)        ;ADDRESS TO HL
01720         RET                     ;DONE
01730 ;
01740 ;       SEARCH FORWARD FOR SPECIFIED 2-BYTE STRING
01750 ;
01760 FSRCH   CALL    PARAM           ;GET VARPTR(MN$)
01770 SEARCH  LD      DE,0000         ;SEARCH STRING
01780         PUSH    HL
01790         POP     IX
01800         LD      C,255           ;STR. POS. COUNTER
01810 FSR010  LD      L,(IX+0)
01820         LD      H,(IX+1)        ;GET FIRST PAIR FOR
CMPR
01830         INC     C
01840         INC     C               ;BUMP STRNG POINTER
01850         BIT     7,H             ;CHECK FOR NEGETIVE
01860         JR      NZ,REVR         ;NEGETIVE - DEAD FILE
01870         RST     18H             ;COMPARE HL/DE
01880         JR      Z,FSR100        ;FOUND
01890         JR      NC,FSR100       ;TARGET>SOURCE
01900 REVR    INC     IX
01910         INC     IX
01920         DEC     B
01930         DJNZ    FSR010          ;LOOP TIL FOUND
01940 FSR100  INC     C               ;STRPOS+1
01950         SRL     C               ;(STRPOS+1)/2
01960         LD      B,0
01970         PUSH    BC
01980         POP     HL
01990         JP      BASIC           ;PASS TO BASIC
02000 ;
```

Listing continued

Listing continued

```
02010 ;         NEW VIDEO DRIVER PATCH
02020 ;
02030 VIDEO     PUSH    AF              ;SAVE FLAGS
02040           PUSH    BC              ;SAVE CHARACTER
02050 VWAIT     IN      A,(USTAT)       ;CHECK UART FOR CLEAR
02060           AND     40H             ;BIT 6 IS THE ONE
02070           JR      Z,VWAIT         ;TRANSMITTER BUSY
02080           LD      A,C             ;GET CHARACTER
02090           OUT     (UDATA),A       ;SEND IT
02100 PF        AND     255             ;LF/CR SWITCH
02110           CP      0DH             ;CAR. RET?
02120           JR      NZ,VID010
02130           LD      C,0AH           ;INSERT L.F.
02140           JR      VWAIT           ;SEND L.F.
02150 VID010    POP     BC              ;RECOVER CHARACTER
02160           POP     AF              ;RECOVER FLAGS
02170 VCONT     JP      0000            ;PATCH POINT
02180 ;
02190 ;         NEW KEYBOARD DRIVER PATCH
02200 ;
02210 KEYBD     PUSH    BC              ;SAVE FLAGS
02220           PUSH    AF
02230           IN      A,(USTAT)       ;CHECK UART
02240           AND     80H             ;CHECK BIT 7 - DATA
REC'D
02250           JR      Z,KOUT          ;NONE READY
02260           IN      A,(UDATA)       ;GET DATA
02270           CP      1               ;CHECK FOR BREAK
02280           JR      Z,KOUT          ;AND IGNORE
02290           LD      C,A
02300           POP     AF              ;RECOVER FLAGS
02310           LD      A,C             ;CHAR. TO A
02320           POP     BC
02330           RET
02340 KOUT      POP     AF
02350           POP     BC
02360 KCONT     JP      0000            ;PATCH POINT
02370 ;
02380 ;         INSTALLATION OF NEW VIDEO AND KEYBOARD
02390 ;         LINKS TO RS-232
02400 ;
02410 ;         FIRST, PATCH VIDEO DCB TO NEW ROUTINE
02420 ;
02430 INSTAL    DI                      ;A LITTLE PRIVACY,
PLEASE
02440           LD      DE,(VBLK+1)     ;GET EXISTING ADDRESS
02450           LD      HL,VIDEO        ;NEW ADDRESS
02460           LD      (VBLK+1),HL     ;PUT NEW ADDRESS IN
DCB
02470           LD      (VCONT+1),DE    ;PUT OLD ADDRESS IN
VIDEO
02480           COND    TRSDOS          ;ASSEMBLE IF TRSDOS
02490 ;
02500 ;         THEN, PATCH KEYBOARD DCB
02510 ;         TRSDOS 1.3 ONLY
02520 ;
02530           LD      DE,(KBLK+1)     ;GET OLD ADDRESS
02540           LD      HL,KEYBD        ;NEW ADDRESS
02550           LD      (KBLK+1),HL     ;INSTALL NEW ADDRESS
02560           LD      (KCONT+1),DE    ;OLD ONE TO OUR
ROUTINE
02570 ;
02580 ;         FINALLY, CONFIGURE UART FOR 300/7/E/1
02590 ;         TRSDOS 1.3 ONLY
02600 ;
02610           OUT     (0E8H),A        ;MASTER RESET
02620           LD      A,55H           ;SET 300 BAUD
SEND/RECV
02630           OUT     (0E9H),A
02640           LD      A,0A4H          ;7/E/1
02650           OUT     (0EAH),A        ;SET IT
02660           ENDC
02670 IN010     EI
02680           JP      TODOS
02690           END     INSTAL
```

End

Program Listing 2. Remaining Basic lines needed to complete the BBS Express.

```
1360 IF (INP(&HE8)AND32)=0 THEN 1380
1365 A$=INKEY$:N=PEEK(&H387F):IFN=0THEN1360
1375 IFA$<>""THENA$=CHR$(ASC(A$)AND95):GOTO6500 ELSE1360
3305 IF (TT$="ALL") AND (SY) THEN TT$=TT$+CHR$(128)
3610    LSET   F1$=NA$:LSET    T2$=TIME$:LSET    F2$=A$:LSET
T1$=TT$:LSET  S1$=S8$:LSET  S2$=CHR$((ASC(S7$)  AND  32)  OR
(ASC(S9$))):PUT 1,SN:PRINT"Awaiting delivery.":GOTO3640
3640 POKE MN,250:RETURN
```

End

everything in the system. Only a wizard or a sysop can change access codes.

Because you don't have a System/BBS data file yet, the BBS has no system defaults or system password. This prevents even a wizard from entering the system, so you'll have to use the break key to get into the BBS this first time. When the system asks for a system password, hit the break key, then type in GOTO 1600 to bypass the log-on procedures.

A Note About Passwords

If you set your system password to PASSWORD, your board becomes an auto sign-on board, meaning that anyone can become a member simply by calling the board. If the system password is something other than PASSWORD, a new caller must know the password before signing on. Under these circumstances, the board is referred to as a closed-access board.

On the first run, the system password could be anything, depending on what was on the disk before you started. You'll be prompted for your name, address, and other personal information. Set your password to something other than PASSWORD—you're the sysop and you must protect your personal BBS records. Set your access level to 0123456789ABCDE*. The asterisk gives you sysop powers.

Next, from the Main Command prompt, enter a dollar sign. This is your point of entry into sysopland and works only if you include an asterisk in your access code. From the Sysop's Access mode, choose S for system, and set your defaults by choosing each of the displayed items by number.

Unless you're running a closed board, we suggest that you set line feeds to Y, video width to 64, and the system password to PASSWORD. The default access is the access level to which the BBS assigns new callers. You can enter digits 0–9 and letters A–E, though you don't have to keep them in order. For example, 03E is acceptable. For starters, set the board's maximum number of messages to 50 to prevent running out of directory slots.

Now choose N from the sysop command menu and name each of the BBS's 15 special-interest sections. Name the sections with care as callers often read special-interest letters only. Use the B command to write a bulletin board message for the new callers, welcoming them to your board. Now you're ready for business.

The message board might cause some problems at first because it requires at least one message to function properly. Hence, you need to set the message index (MB$). To do this, type in a dollar sign ($) at the sysop command to break the program. Now, in the immediate mode, enter MB$ = MKI$(1) + STRING$(98,CHR$(0)). This indexes the very first message, though you still need a header. To get one, type in the following:

```
GOSUB 220:GET 1,1:LSET T1$="ALL"
:LSET F1$="SYSOP":LSET F2$="MSG
0001":LSET S2$=CHR$(1):LSET S1$=
"WELCOME":LSET T2$=TIME$:PUT1,
1:CLOSE.
```

Next, you need to set up the System/BBS file. The command for this is:

```
SH=1:SN=1:SL=1:GOSUB190:GET
3,1:LSET SN$=MKI$(SN):LSET SL$=
MKI$(SL):LSET SH$=MKI$(SH):LSET
SC$=MKI$(1):LSET NM$=MKI$(1):LSET
DS$=MKI$(0):LSET ND$=MKI$(0):PUT3,
1:CLOSE
```

Reenter the program now by typing in CONT. We have one message and one header indexed, so write a message and log off with the E command. Never exit without first logging off with this E command.

An easier way to handle the initial business of setting up your board is to call us at 606-739-6088 and pick up a copy of INTRO/BAS from our data base—it does all the above for you.

Closing Thoughts

Program Listing 1 is the machine-language assist module; Program Listing 2 provides a few lines we missed in previous listings. You should have no problems incorporating these listings with those from other BBS Express installments.

The COND and ENDC statements in Listing 1 are conditionals. If your assembler doesn't support this syntax, delete the lines marked for TRSDOS to make an LDOS version.

Originally, we thought that the BBS Express was relatively crash proof. In November, however, we received reports of repeated crashes and realized that the vandals were waiting out the modem and gaining access to the title page options. Lines 2705 and 1335 of Listing 2 prevent this from happening.

TRSDOS seems to be more of a problem than we originally anticipated, particularly with regard to the sysop Submit command. One version handles variable length records poorly, while the other doesn't handle them at all. In addition, TRSDOS allows only 80 files on a data disk, and if the total exceeds 80, TRSDOS reports that the disk is full. If you have a high-traffic operation in mind, drop us a line and we'll fix you up with a high-volume version of the BBS Express. LDOS users, on the other hand, report no problems. In fact, if you switch to LDOS, you get 112 files, a 40 percent increase in board capacity.

We've had a lot of fun with the BBS Express and hope it has been a rewarding experience for all of you who've been following us for the past year. ■

This installment marks the end of the BBS Express, 80 Micro*'s do-it-yourself bulletin board. To see the finished product, call the* 80 Micro *BBS at 603-924-6985. UART parameters are 300 baud, 7-bit words, 1 stop bit, and even parity.*

You can reach J. Stewart Schneider and Charles E. Bowen either through their bulletin board at 606-739-6088 or c/o Saturday Software, P.O. Box 404, Catlettsburg, KY 41129.