

Exchange

September 1985

Software

- 1** IBM BASIC Compiler 2.00 Overview (Part1)
- 7** A Review of IBM PC Storyboard
- 9** ANSI.SYS ESCape Codes
- 13** TopView Questions and Answers (Part 1)
- 18** DOS Device Drivers (Part 1)
- 22** Customize Your DOS Prompt

Getting Started

- 23** DOS Filter Commands
- 26** Watch Where You Copy To!

Random Data

- 27** Defining the Upper 128 ASCII Characters
- 30** Robot Simulation in BASIC
- 31** Keyboard Input for BATch Files
- 32** IBM PC AT Serial Port Pin Configuration
- 33** Puzzler

Departments

- 34** New Products
- 36** Editor's Comments

Exchange of IBM PC Information



Exchange of IBM PC Information is a monthly publication of the National Distribution Division, International Business Machines Corporation, Boca Raton, Florida, USA.

Editor	Michael Engelberg
User Group Editor	Bernard Penney
Associate Editor,	
Design Director	Karen Porterfield
Writer	John Warnock
Editorial Assistant	Steve Mahlum
Illustrators	Michael Bartalos
	John Alfred Dorn III
	Narda Lebo
	John Segal
	Charles Slackman
Production	Cohen and Company
User Group	
Support Manager	Gene Barlow

Exchange of IBM PC Information is distributed at no charge to registered PC user groups. To register with us, please write to:

IBM PC User Group Support
IBM Corporation (2900)
P.O. Box 3022
Boca Raton, FL 33431-0922

To correspond with *Exchange*, please write to:

Editor, *Exchange*
IBM Corporation (2900)
P.O. Box 3022
Boca Raton, FL 33431-0922

POSTMASTER: send address changes to *Exchange of IBM PC Information*, IBM Corporation (2900), P.O. Box 3022, Boca Raton FL 33431-0922.

IBM cannot be responsible for the security of material considered by other firms to be of a confidential or proprietary nature. Such information should not be made available to IBM.

IBM has tested the programs contained in this publication. However, IBM does not guarantee that the programs contain no errors.

IBM hereby disclaims all warranties as to materials and workmanship, either expressed or implied including without limitation, any implied warranty of merchantability or fitness for a particular purpose. In no event will IBM be liable to you for any damages, including any lost profits, lost savings or other incidental or consequential damage arising out of the use or inability to use any information provided through this service even if IBM has been advised of the possibility of such damages, or for any claim by any other party.

Some states do not allow the limitation or exclusion of liability for incidental or consequential damages so the above limitation or exclusion may not apply to you.

It is possible that the material in this publication may contain reference to, or information about, IBM products, programming or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming or services in your country.

An Overview of IBM BASIC Compiler 2.00

(Part 1)

Glenn Crumpley
IBM Corporation

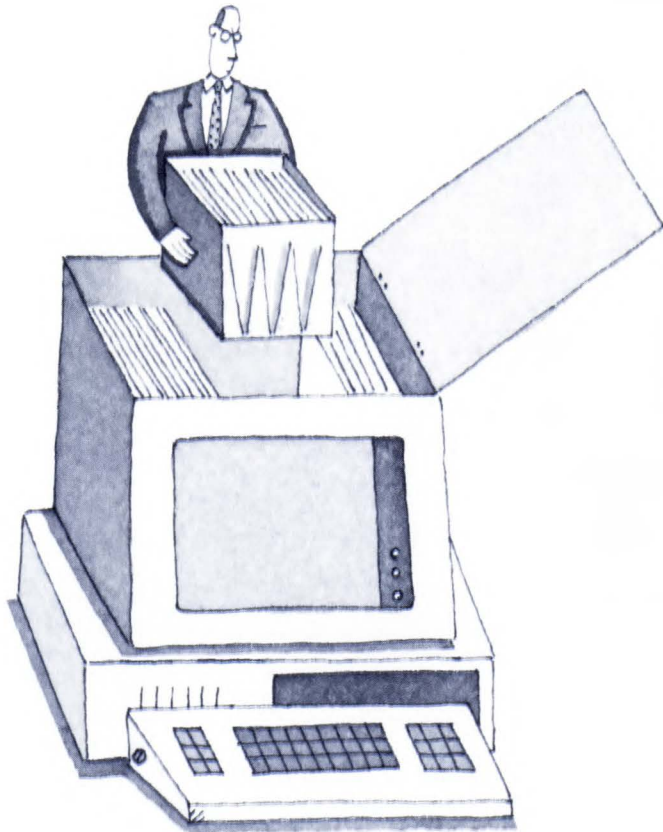
Editor's note: This is the first part of an article on IBM BASIC Compiler 2.00, covering the new features of version 2.00 and differences between the Compiler and Interpreter. Part 2, covering modular programming techniques and using ISAM files, will be published next month. This article is adapted from the IBM Personal Computer Seminar Proceedings.

The BASIC Compiler 2.00 is an optimizing compiler designed to complement the BASIC Interpreter. (Optimizing compilers do such things as change the order of expressions or eliminate common sub-expressions to either improve performance or decrease the size of the programs.)

Creating application programs with the IBM Personal Computer BASIC Compiler 2.00 provides several benefits, some of which are:

- Networking support (Lock and Unlock statements)
- Full graphics support
- Full PCjr compatibility
- Increased file capacity (16,775,616 record maximum—formerly 32,767)
- Increased input string length (32,767 character maximum—formerly 255 character maximum)
- Full DOS file capability
- Shell commands support
- Line numbers unnecessary
- Separately compiled modules allow creation of larger programs
- Increased execution speed for most programs when compared to the interpreter version
- BASIC source code security

The BASIC Compiler 2.00 offers a powerful programming environment in which you can use the BASIC Interpreter to quickly run and debug programs and then later compile those programs to increase their execution speed.



A compiled program is optimized machine code, not source code. Consequently, compiling substantially improves execution time and protects your source program from unauthorized alteration or disclosure.

Licensing Agreement

Application programs that require the runtime modules BASRUN20.EXE, REBUILD.EXE, or ISAM.EXE cannot be distributed without entering into a license agreement with IBM. A copy of the license agreement can be obtained by writing to:

IBM Corporation
P.O. Box 2910
Delray Beach, Florida 33444
Attn: Personal Computer Customer Relations

Note, however, that by compiling with the /O parameter, it is possible to develop programs with the BASIC Compiler 2.00 that do not use the BASRUN20.EXE runtime module and, therefore, do not require the license agreement. This does not apply to ISAM.EXE or REBUILD.EXE.

Hardware Requirements

The hardware necessary to use this product is:

- Any of the following IBM Personal Computers:
 - IBM Personal Computer
 - IBM Personal Computer XT
 - IBM Personal Computer AT
 - IBM Portable Personal Computer
 - IBM PCjr
- A minimum of 128KB of Random Access Memory (RAM) (Additional memory can significantly improve the performance of the BASIC Compiler 2.00 and the Linker when used on all of the above listed computers.)
- One or two double-sided diskette drives or a fixed disk
- A printer (highly recommended)
- A display screen (Although various displays can be used, best results are obtained with an 80-column display.)
- Blank, formatted diskettes

Software Requirements

The software necessary to use this product is:

- Disk Operating System (DOS) 2.10 or later version

Changes in BASIC Compiler 2.00

BASIC Compiler 2.00 differs from BASIC Compiler 1.00 in the following areas:

- Improved program control structures allow a more modular approach to programming. Enhancements include:

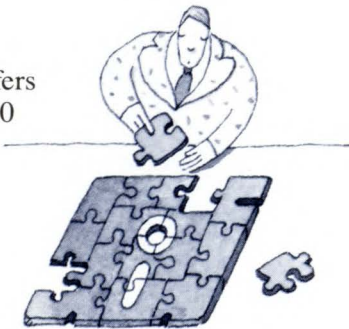
Named subprograms—provides the ability to call (execute) a routine or subprogram by a name instead of a line number.

Named COMMON blocks—can be used for inter-module communication without chaining. Items listed in blank COMMON can be accessed by another chain file.

User-defined multiline functions—permits a function definition to occupy more than one program line. It must begin with a DEF FN statement and end with an END DEF statement.

Ability to branch to alphanumeric labels—it is no longer necessary to use only line numbers; now meaningful statement labels may be used (example: GOTO TOTALS).

Separately compiled BASIC subprograms.



- Larger programs can be compiled. The use of a memory model that separates the instruction space from the data space allows this, as well as allowing more than twice as much symbol table space. Please note, however, that the data segment has a maximum upper limit of 64KB. In addition, allocated string space is also limited to a maximum of 64KB.
- Large dynamic arrays are supported. The maximum index for any dimension of a numeric array is 32767. This dimension limit and the amount of memory in your machine are the only size restrictions for numeric arrays.
- .EXE files produced by BASIC Compiler 2.00 are larger than those produced by BASIC Compiler 1.00.
- Graphics capabilities are expanded. All graphics features of the BASIC Interpreter are available. These include:

VIEW	DRAW
WINDOW	POINT
PMAP	PAINT
LINE	

- Access to DOS is expanded. Several new features of the BASIC Interpreter are available to allow more flexible use of DOS functions. Statements affected are:

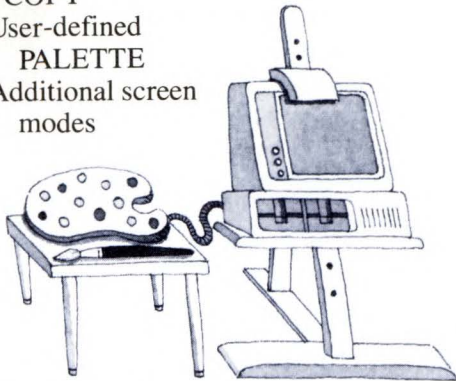
SHELL	ERDEV
IOCTL	ERDEV\$
IOCTL\$	MKDIR
ENVIRON	RMDIR
ENVIRON\$	CHDIR

- The *filespec* syntax is expanded to allow the specification of a path for a device or file.
- Redirection of standard input and standard output is supported.
- Enhanced event trapping is available. This enhancement affects the following statements:

ON TIMER
ON PLAY
ON KEY

- All advanced features of PCjr BASIC are supported. The full range of sound and graphics capabilities are available to users of PCjr. Some of the features include:

PLAY—Multi-voice
 PLAY—Volume Control
 NOISE
 Enhanced SCREEN statement
 Enhanced CLEAR statement
 PCOPY
 User-defined
 PALETTE
 Additional screen
 modes



- Compiler termination codes are returned when the compiler exits. These termination codes can be tested by the IF batch subcommand of DOS.
- An input editor is included. Input required by your program can be altered easily on the screen.
- Support is provided for up to five levels of nested \$INCLUDE files.
- When compiling, you must specify the /D parameter to Ctrl-Break effectively at runtime.
- BASIC library files are searched in the following order:
 1. User-specified directory
 2. Current directory
 3. PATH directories
 4. User-prompted directory.
- Graphics statements now use line clipping instead of wraparound.
- The OPEN statement has been enhanced to include file access control.
- Because of the added functions in BASIC Compiler 2.00, you may notice slightly longer compile and link times.

BASIC Compiler 2.00 includes the following language additions:

- **Statements**

- CALLS**

- Calls and transfers program control to IBM Personal Computer Macro Assembler routines.

- DEF FN, END DEF, EXIT DEF**

- Designate the beginning and ending of a multiline function.

- LOCK, UNLOCK**

- Restrict access by other processes to all or part of an opened file.

- REDIM**

- Changes the space allocated to a dynamic array.

- SHARED**

- Designates variables as global to the subprogram and the calling program.

- STATIC**

- Designates variables as local to a subprogram or multiline function.

- SUB, END SUB, EXIT SUB**

- Designate the beginning and ending of a subprogram.

- **Functions**

- COMMAND\$**

- Returns the parameters from the command line used to invoke the current program.

- LBOUND**

- Returns the value of the lowest subscript available (either 0 or 1) for any array. This value depends on the setting of the OPTION BASE statement.

- UBOUND**

- Returns the value of the largest subscript available for any array.

- **New File Type — ISAM**

- The BASIC Compiler 2.00 now supports the indexed sequential access method. These ISAM files are accessed through the CALL statement. ISAM files allow for rapid access to large files by key values. Other features are automatic storage space management and fast sequential access.

- **Library Manager**

- The IBM Library Manager is included. This utility enables you to construct and edit object module libraries. See Appendix E of the *BASIC Compiler Fundamentals* manual for details.

Differences Between the Compiler and Interpreter

Differences between the languages supported by the BASIC Compiler 2.00 and the BASIC Interpreter must be taken into account when compiling existing or new BASIC programs.

The differences between the languages supported by the BASIC Compiler 2.00 and the BASIC Interpreter are described below:

Operational Differences

Some BASIC commands and statements used to operate in the interpreter programming environment are not acceptable input to the compiler. These are:

AUTO	LOAD
CONT	MERGE
DELETE	NEW
EDIT	RENUM
LIST	SAVE
LLIST	

Certain statements function similarly in the BASIC Compiler 2.00 and the interpreter, but require special parameters to be specified when used with the compiler.

- **Event trapping:** If you use any of the event trapping statements, you must specify either the /V or the /W parameter when you start the compiler. The event trapping statements are:

COM(n)	ON STRIG(n)
KEY(n)	ON TIMER
ON COM(n)	PEN STOP
ON PEN	PLAY(n)
ON PLAY	STRIG(n)

- **Error trapping:** If you use an ON ERROR statement and some form of a RESUME statement, you must specify either the /E or the /X parameter when you start the compiler. If you use only the RESUME *line* form, you should specify /E. If you use RESUME NEXT, RESUME 0, RESUME, or any combination of those with RESUME *line*, the /X parameter must be used instead.
- **Debug code (TRON and TROFF):** To use TRON and TROFF, the /D parameter must be specified when you run the compiler. Otherwise, TRON and TROFF are ignored and a warning is generated.

Note that using these parameters increases the size of the .OBJ, and .EXE files. See the *BASIC Compiler Fundamentals* for a detailed explanation of each of the compiler parameters.

Language Differences

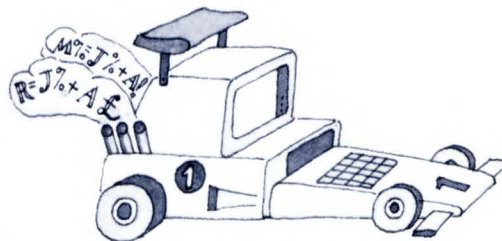
If your machine has a cassette port, the BASIC Compiler 2.00 supports cassette I/O. However, to enable cassette I/O, you must specify the /O parameter at compile time and then link the IBMCAS.OBJ module.

Some differences exist among the commands, statements, and functions of the BASIC Compiler 2.00 and BASIC Interpreter. These differences are explained in the *BASIC Compiler Language Reference* manual.

Other Differences

Other differences between the BASIC Interpreter and the BASIC Compiler 2.00 include the following:

- **Double-Precision Arithmetic Functions**
If you use double-precision operands for any of the arithmetic functions, including the transcendental functions (SIN, COS, TAN, ATN, LOG, EXP, and SQR), the BASIC Compiler 2.00 returns double-precision results. In the interpreter, double-precision results are returned if the interpreter is invoked with the /D parameter.
- **Double-Precision Loop Control Variables**
Unlike the interpreter, the compiler allows the use of double-precision loop control variables. This allows you to increase the precision of increment in loops.
- **Expression Evaluation**
Mathematical computations have been modified in the compiler for improved speed and accuracy, so there may be slight differences in the results of single-precision or double-precision operations compared to the interpreter.



Also, the BASIC Compiler 2.00 performs optimization, if possible, when evaluating expressions.

During expression evaluation, the BASIC Compiler 2.00 converts operands of different types to the type of the more precise operand.

$$QR = J\% + A! + Q\#$$

The above expression causes J% to be converted to single-precision and added to A!. This double-precision result is added to Q#.

The BASIC Compiler 2.00 is more limited than the interpreter in handling numeric overflow. For example, when run on the interpreter, the following statements yield 40000 for M.

```
I% = 20000
J% = 20000
M = I% + J%
```

That is, J% is added to I%. Because the number exceeds the 32767 limit for integers, the interpreter converts the result into a floating-point number. The result, 40000, is found and saved as the single-precision number M.

The BASIC Compiler 2.00, however, must make type conversion decisions during compilation. It cannot defer until actual values are known. Thus, the compiler generates code to perform the entire operation in integer mode and arithmetic overflow may occur. If the /D debug parameter is set, the error is detected. Otherwise, an incorrect answer is produced. One possible way to avoid this problem is to use single-precision numbers instead of integers.

Besides the previous type conversion decisions, the compiler performs certain valid optimizing algebraic transformations before generating code. For example, the following program could produce an incorrect result when run:

```
I% = 20000
J% = -18000
K% = 20000
M% = I% + J% + K%
```

If the compiler actually performs the arithmetic in the order shown, no overflow occurs.

However, if the compiler performs $I\% + K\%$ first and then adds $J\%$, overflow occurs. The compiler follows the rules of operator precedence, and parentheses may be used to direct the order of evaluation. No other guarantee of evaluation order can be made.

• Input Statements

The compiler limits the number of variables read by an INPUT or INPUT # statement to 60.

If you try to enter more than 32767 characters in response to any INPUT or LINE INPUT statement, the compiler makes the computer sound a beep.

• Integer Variables

The BASIC Compiler 2.00 can make optimum use of integer variables as loop control variables. To help the compiler produce faster and more compact object code, you should use integer variables as much as possible. For example, the following program executes much faster by replacing I, the loop control variable, with I%, or by declaring I an integer variable with DEFINT.

```
100 FOR I = 1 TO 10
110 A(I) = 0
120 NEXT I
```

It is also advantageous to use integer variables to compute array subscripts because the generated code is faster and more compact.

Input Editor

When you respond to an input statement in a compiled program, you do not have all the facilities of the BASIC program editor to use. The BASIC Compiler 2.00 does not allow you to change lines anywhere on the screen; you may edit only the current line.

The input editor supplied with BASIC Compiler 2.00 uses a special set of commands to manipulate the text on the screen. These commands are different from the commands used by the editor in the BASIC Interpreter.

Input Editor Commands: All of the editor commands, except Delete, require you to press the control key (Ctrl) in combination with another key.

Ctrl-B	moves cursor back one word.
Ctrl-C	exits program.
Ctrl-E	erases to the end of the current line.
Ctrl-F	moves cursor forward one word.
Ctrl-H	deletes the character to the left of the cursor.
Ctrl-I	inserts spaces from the cursor position up to the next tab position (tabs are set every eight spaces). If the editor is in replace mode, any existing characters will be overwritten.
Ctrl-K	moves the cursor to the beginning of the line.
Ctrl-M	issues a carriage return and enters the line.
Ctrl-N	moves the cursor to the end of the line.
Ctrl-R	toggles the editor from insert to replace mode.
Ctrl-T	toggles the function key display line on and off.
Ctrl-U	erases the entire line.
Ctrl-]	moves the cursor one position to the left.
Ctrl-\	moves the cursor one position to the right.
Del	deletes the character at the cursor.

The following special program editor keys are not supported by the compiler:

```
Home
Ctrl-Home
Cursor Up
Cursor Down
Next Word (Ctrl-Cursor Right)
Previous Word (Ctrl-Cursor Left)
Ctrl-Break.
```

If you try to use any of these keys (with the exception of Ctrl-Break) in response to an input statement, the computer will sound a two-tone beep.

Pressing Ctrl-Break in response to an input statement returns you to DOS.

All files are closed and the following message is displayed:

```
STOP in Line xxx of Module
Modulename at Address ---:---
```

Hit any key to return to system

When a key is pressed, the DOS screen mode is restored.

Number of Files

The maximum number of files that can be open simultaneously is 15. The default value is 3. To increase the number of files to be opened simultaneously you must have the following in your CONFIG.SYS file:

```
FILES=xxx
```

where:

xxx is the number of files you plan to have open simultaneously, plus 5, which are used by DOS. The maximum value for xxx is 20.

Line Length

The interpreter cannot accept lines greater than 254 characters in length. In contrast to the interpreter, the BASIC Compiler 2.00 accepts *physical* lines of up to 32766 characters in length. (A *physical* line for the compiler is one that ends in a carriage return-line feed.) However, you can make the compiler accept much longer *logical* lines of input by ending the physical lines with an underscore character (underscores in quoted strings or remarks do not count). The underscore tells the compiler to ignore the following carriage return, so all it sees in the carriage return-line feed sequence at the end of the line is the line feed character. The line feed is the line continuation character understood by the compiler. For example, the following two physical lines:

```
100 INPUT "Values for array A"; A(1),_
A(2), A(3), A(4), A(5), A(6), A(7)
```

are read by the compiler as a single INPUT statement that enters seven values into array A.

It is impractical to use this technique with the program editor in the BASIC Interpreter because each line created with the BASIC Interpreter editor must begin with a number. In addition, BASIC Compiler 2.00 source programs that use this technique cannot be debugged using the interpreter.

PEEKs and POKEs

PEEKs and POKEs into the interpreter work area (such as DEF SEG: POKE 106,0) are interpreter dependent and do not work for compiled BASIC.

Note: PEEK and POKE for dynamic array elements work differently than PEEK and POKE for static array elements. See the PEEK and POKE statements in *BASIC Compiler 2.00 Language Reference* for details.

String Length

Strings can be up to 32,767 characters long rather than 255 characters long. Therefore, any string function parameters that identify the location in a string or its length (which can have a maximum value of 255 in the interpreter) can now range to 32,767.

The internal storage format for the string descriptor requires four bytes rather than three bytes (low byte, high byte of the length, followed by low byte, high byte of the address). If you use machine language subroutines with string arguments, you have to recode the subroutine to account for this change.

String Space Implementation

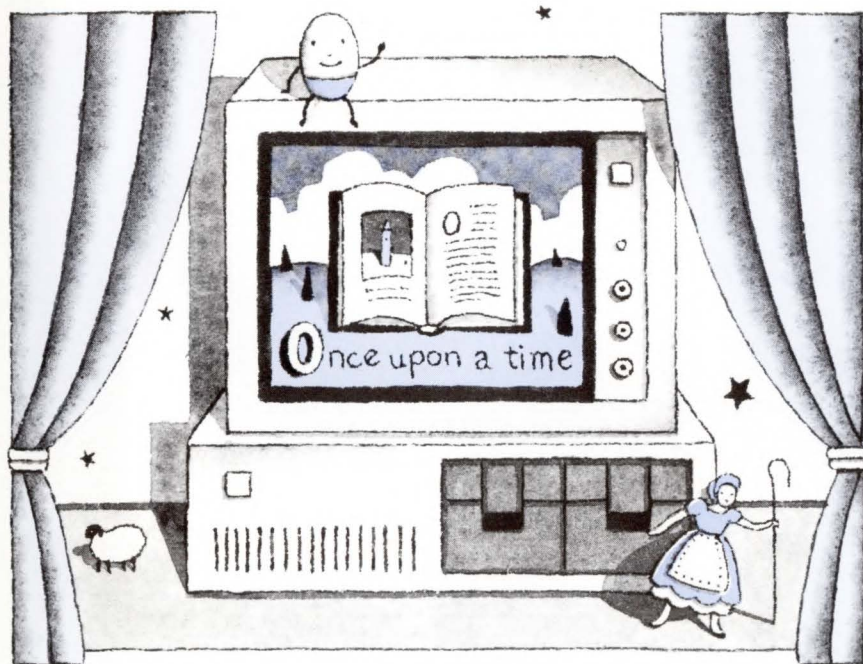
The implementation of the string space for the compiler differs from its implementation for the interpreter. Using PEEK, POKE, VARPTR or assembly language routines to change string descriptors may result in a String Space Corrupt error.

The amount of available string space is 64KB. Because of the string descriptor size and the number of internal variables used by the compiler, the number of elements in a string array is less than the amount available in BASIC Compiler 1.00 or the BASIC Interpreter. To help maximize the use of memory space, you can move numeric data into dynamic arrays that have a separate data area. Declare string arrays as dynamic and ERASE (reuse) the same space whenever possible. You also can use MID\$ to help prevent fragmentation of string space. See the example under "MID\$ Function and Statement" in the *BASIC Compiler 2.00 Language Reference* for details.

A Review of IBM PC Storyboard

Bud Thurber

North East Indiana IBM PC Club



I recently had the opportunity to use one of the most delightful and refreshing new business programs that I have seen in months. It is IBM's new PC Storyboard. It is almost as much fun to use as gameware, but the resulting output is seriously useful in business. Business means making sales and making sales means making presentations. That's where Storyboard comes in. It assists a salesperson in making presentations, even presentations that run all by themselves.

Most business computer software falls into just a few categories: spreadsheets, word

processors, file managers, and accounting packages. We find our favorites in each of these categories, something that gives us a nice balance between "easy to use" and "sophisticated and powerful," and we tend to stick with these favorites even when something better comes out. But the information that these other applications generate often has to be integrated into presentations. For those who make presentations, PC Storyboard offers four powerful programs that help you create colorful and interesting displays.

Storyboard allows you to create a series of displays or "pictures" on the PC screen. It uses a very

high level language, complete with DISPLAY, GOTO, GOSUB, and other commands. But you don't need to know the language. It is included as part of the program called "Story Editor," which allows you to select "pictures" stored on disk, put them in any order, and then use various methods of presenting them, one picture at a time. These methods include vertical, horizontal, and diagonal screen wipes, plus dissolves, explosions, etc. With a little artistic endeavor, you can even do limited animation.

The individual pictures are constructed using another powerful program called "Picture Maker." Each picture can include text, graphs and charts, or objects such as people, cars or buildings. You can include any combination of these entities in each picture, move them around, size them, and specify what color each will be.

You select text from several fonts already included with PC Storyboard (Bold, Roman, and others), and each font can be displayed in several sizes from tiny to giant letters. You can then add shadowing and color. If you don't see the font you want, you can create your own.

The graphs and charts include vertical and horizontal bar charts, line graphs, and pie charts, which are easily constructed using your own values and a few commands.


PC Storyboard includes several libraries of symbols and objects including arrows, pointing hands, people, factories, and even a whole city. You display the library screen with the object you want,

“cut” the object from that screen and then “paste” it on your own picture (cut and paste). Using a ZOOM command, you can resize the object and then locate it anywhere in the picture.

Perhaps you would rather use original material from spreadsheet screens or original graphs and pictures created by some other program such as Lotus 1-2-3, or other language such as BASIC or Pascal. You can do this with another Storyboard program, “Picture Taker,” a DOS-resident program that replaces the Print Screen function. After loading Picture Taker, you simply run your program until the material you want in your presentation appears on the screen. Then you press Shift-PrtSc; instead of copying the screen to the printer, the screen image is copied to a disk file. Each screen image or picture captured in the disk file can then be further modified and enhanced with the Picture Maker program or just included as one picture in the presentation you create using Story Editor.

For example, you could produce a graph with your accounting program, save it using Picture Taker, label it “Factory Output,” and then, using Picture Maker, paste a picture of a little factory on your graph.

Once you have captured screen images using Picture Taker, created and modified your individual pictures with Picture Maker, and edited them into a story with Story Editor, you will use the fourth program included with PC Storyboard—Story Teller—to actually present your story. Everything you need to tell a story (or make a sales presentation) can then be put on a single diskette, including DOS, the pictures that make up your presentation, and Story Teller.



IBM PC Storyboard
*could renew home interest
in PCs. It could turn into a
whole new fun-and-profit
hobby all by itself.*

Story Teller can tell your story in several ways: (1) Each picture can be put on the screen for a specified period of time and then automatically advanced to the next picture. (2) You can specify that some or all pictures will wait until a keyboard key is pressed. (3) You can use the game adapter and the “fire” buttons on a joystick. The “A” button will advance your story and the “B” button will back up just like the remote buttons on a slide projector.

The Story Teller program can be copied onto your distribution diskette without royalties as long as you follow the guidelines that are explained in the manual, such as putting the proper IBM copyright notices on each presentation diskette.

IBM PC Storyboard can be used at home. Consider the possibilities:

- Loads of fun for you and the kids as you create various stories and presentations.
- You could write additional cut-and-paste picture files or text fonts and sell them through share-ware (similar to selling spreadsheet templates).
- Make up your own presentations at home for use at your business.
- Make classroom presentations.
- Make and sell story diskette/ audio cassette packages for canned presentations or “talking books.”
- Get a bunch of PCjrs and color TVs and rent them to store owners along with customized video stories announcing current sales items in their stores. This kind of story can be told in an unattended program loop.

IBM PC Storyboard could renew home interest in PCs. It could turn into a whole new fun-and-profit hobby all by itself.

ANSI.SYS ESCape Codes

Steven Mahlum
IBM Corporation

ANSI.SYS is a device driver that comes with DOS 2.00 and higher. It interprets special character sequences (ESCape codes), using them to reassign keyboard key definitions, control the position of the cursor, and set the mode of operation, such as screen attributes.

To use the ANSI.SYS commands, you must first load the ANSI.SYS device driver by adding the following line to your CONFIG.SYS file:

```
DEVICE=ANSI.SYS
```

When you start your computer with this command in the CONFIG.SYS file, DOS loads ANSI.SYS into memory. Be sure you have the ANSI.SYS file on your DOS diskette, or, if you are starting from a fixed disk, specify the path where DOS can find the ANSI.SYS file. For example,

```
DEVICE=C:\DOS\ANSI.SYS
```

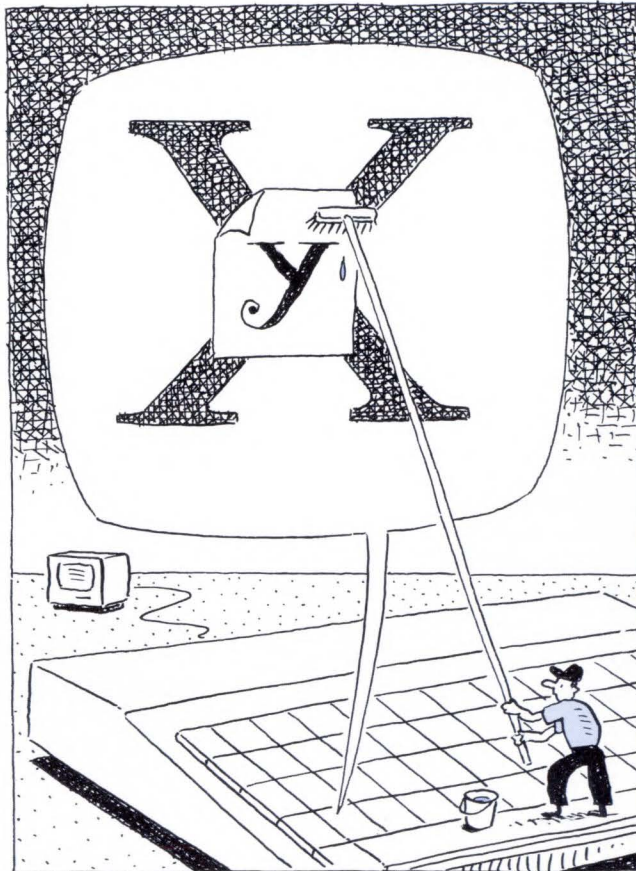
If you do not have a CONFIG.SYS file in the root directory of your DOS disk(ette), create one and put this command in it. (You may want other commands in your CONFIG.SYS file. See your DOS Reference Manual or review R.K. Schmitt's article "The DOS CONFIG.SYS File" in the August 1985 issue of *Exchange*.)

To use the ANSI.SYS ESCape codes, the ESCape character (ASCII 27) must start the code sequence. In this article, the characters ESC represent the ASCII 27 ESCape character.

To create these codes, you can use an ASCII text editor such as IBM Personal Editor, EDLIN, etc., to enter the ESCape codes directly into a text file or BATch file. You also can use the DOS PROMPT command with the predefined \$e macro to represent the ESCape character. Using the PROMPT command to enter ANSI.SYS ESCape codes eliminates the need for a text editor that allows ASCII 27 to be entered from the keyboard; you can use the DOS COPY command to create files with ESCape codes directly from the keyboard.

To take effect, the ANSI.SYS ESCape codes must be sent to the system output device, usually the display. To do this, you can:

- Enter the ESCape codes in an ASCII text file and use the DOS TYPE command to send them to the display.



- Imbed the ESCape codes in a BATch file and begin each line containing ANSI.SYS ESCape codes with either the REM or ECHO command. You can send the codes to the display by executing the BATch file or using the DOS TYPE command.
- Use the DOS PROMPT command with the \$e macro. You can enter the PROMPT command directly from the DOS command line or include it in a BATch file. However, you must run the BATch file to send the ESCape codes to the system output device. ESCape codes in the PROMPT format will not take effect when TYPED to the screen.

For a concise listing of all ANSI.SYS ESCape codes, see the pull-out chart at the center of this issue.

Keyboard Key Reassignment Codes

ANSI.SYS allows you to reassign the keyboard key definitions. You can reassign a key to issue other ASCII values, including character strings. The format for entering key reassignment codes is:

```
ESC[#;...;#p
```


where the first # parameter in the sequence is an ASCII value specifying which key definition is to be reassigned. The succeeding # parameters define the sequence of ASCII values and character strings that will be generated when the specified key is intercepted. The key reassignment code must always end with p. For example, to reassign A to become Q, enter

```
ESC[65;81p
```

or

```
PROMPT $e[65;81p
```

To reassign an extended ASCII code, the first # parameter must be 0 (null) and the second # parameter will give the extended ASCII value. For example, to reassign the F10 key to display the directory information, enter

```
ESC[0;68;"dir";13p
```

or

```
PROMPT $e[0;68;"dir";13p
```

To execute a carriage return as part of the new definition (i.e., to execute a command without having to press Enter), be sure that the ASCII value 13 is the last parameter entered.

To reassign the F9 key to display the DOS TYPE command, enter

```
ESC[0;67;"TYPE "p
```

or

```
PROMPT $e[0;67;"TYPE "p
```

This causes the TYPE command to appear on the DOS command line, waiting for you to enter a file name and press the Enter key.

Figure 1 contains a list of extended ASCII codes, and the key assigned to each code. You may want to reassign those key definitions. However, you could reassign your whole keyboard if you wish to experiment with a different style keyboard, such as the Dvorak keyboard.

Extended ASCII Codes	
Extended ASCII Code	Definition
3	NUL (null character)
15	Shift tab
16-25	Alt—Q, W, E, R, T, Y, U, I, O, P
30-38	Alt—A, S, D, F, G, H, J, K, L
44-50	Alt—Z, X, C, V, B, N, M
59-68	F1, F2, F3, F4, F5, F6, F7, F8, F9, F10
71	Home
72	Cursor Up
73	Page Up
75	Cursor left
77	Cursor right
79	End
80	Cursor down
81	Page Down
82	Insert
83	Delete
84-93	Shift—F1, F2, F3, F4, F5, F6, F7, F8, F9, F10
94-103	Ctrl—F1, F2, F3, F4, F5, F6, F7, F8, F9, F10
104-113	Alt—F1, F2, F3, F4, F5, F6, F7, F8, F9, F10
114	Ctrl—PrtSc
115	Ctrl—Cursor left
116	Ctrl—End
117	Ctrl—Cursor down
118	Ctrl—Ins
119	Ctrl—Home
120-131	Alt—1, 2, 3, 4, 5, 6, 7, 8, 9, 0, -, =
132	Ctrl—PageUp

Figure 1. Extended ASCII Codes

You will probably want to build a BATch file containing all your key reassignments that you can either run or TYPE to the screen.

The following ESCape codes reassign the Alt + function keys.

```
ESC[0;113;"dir c:";13p
ESC[0;112;"dir b:";13p
ESC[0;111;"dir a:";13p
ESC[0;110;"dir /w";13p
ESC[0;109;"chkdsk";13p
ESC[0;108;"exit";13p
ESC[0;107;"type "p
ESC[0;106;"diskcopy a: b:";13p
ESC[0;105;"copy *.* b:";13p
ESC[0;104;">prn:";13p
```


When loaded, some programs do not return reassigned keys to their default definitions. If you need to return the keys to their default definitions, enter the following in another BATch file.

```
ESC[0;113;0;113p
ESC[0;112;0;112p
ESC[0;111;0;111p
```

You can create a file to effect key reassignments directly from the keyboard using the COPY command and the PROMPT command. To create a file, MYKEY.BAT, from the keyboard that has all the redefinitions shown above, type the following at the DOS prompt. <CR> means press the Enter key; <F6> means press the F6 key.

```
COPY CON: A:\MYKEY.BAT <CR>
PROMPT $e[0;113;"dir c:";13p <CR>
PROMPT $e[0;112;"dir b:";13p <CR>
PROMPT $e[0;111;"dir a:";13p <CR>
PROMPT $e[0;110;"dir /w";13p <CR>
PROMPT $e[0;109;"chkdsk";13p <CR>
PROMPT $e[0;108;"exit" 13p <CR>
PROMPT $e[0;107;"type "p <CR>
PROMPT $e[0;106;"diskcopy a: b:";13p <CR>
PROMPT $e[0;105;"copy *.* b:";13p <CR>
PROMPT $e[0;104;">prn:";13p <CR>
PROMPT $P$G <CR>
<F6> <CR>
```

When you use the PROMPT command to enter key definitions, be sure that the last PROMPT command will give you the DOS prompt you want displayed on your screen.

Using the ANSI.SYS ESCape codes, you can temporarily reassign a key to enter a specific character string that you will have to type often in a document. If, for example, the phrase "TopView environment" occurs frequently, you could reassign the F10 key (or some other key) to enter the phrase whenever it is pressed.

```
ESC[0;68;"TopView environment "p
```

(This type of reassignment works only with those programs that do not redefine the function keys when loaded.)

Cursor Control Codes

ANSI.SYS allows you to control the position of the cursor on the screen. The following ESCape codes control the cursor position:

- ESC[*#*;*#*H Specifies the cursor position by row and number. For example, to place the cursor at row 1, column 20, you could enter the following in a text file:

```
ESC[1,20H
```

and TYPE it to the screen, or you could enter it in a BATch file as

```
ECHO ESC[1,20H
```

or

```
PROMPT $e[1,20H
```

- ESC[*#*A Moves the cursor up *#* rows from its current position. The default value is 1.
- ESC[*#*B Moves the cursor down *#* rows from its current position. The default value is 1.
- ESC[*#*C Moves the cursor forward *#* columns from its current position. The default value is 1.
- ESC[*#*D Moves the cursor backward *#* columns from its current position. The default value is 1.
- ESC[*#*;*#*f Specifies cursor position by line and column number. The default value for each parameter is 1. If no parameter is given, the cursor moves to the home position.
- ESC[*#*;*#*R Reports the current cursor position through the standard input device
- ESC[6n Causes the console driver to output a cursor position report when a request for device status report is received. If this command is entered using the PROMPT command, it must not be the last PROMPT command given in the BATch file or it will cause a continuous loop, forcing you to restart your computer.
- ESC[s Saves the current cursor position. It can be restored using ESC[u.
- ESC[u Restores the cursor to the position last saved with ESC[s.
- ESC[2J Erases the screen and moves the cursor to the home position.
- ESC[K Erases from the cursor position to the end of the line (including the cursor position).

The cursor control codes allow you to create some interesting variations of your DOS prompt. Below are a few examples of customized prompts using the ANSI.SYS cursor control codes together with the predefined PROMPT macros.

To have the time appear at the far right of the line just above the DOS command line and a prompt that displays the default drive and current directory, enter

```
PROMPT $e[s$e[1A$e[75C$t$h$h$h$h$h$h$
      $h$e[u$p$g
```

If you were in the BIN directory on drive C, the prompt would appear:

```
C:\BIN >
```

18:35

To create a prompt that displays the current directory on the default drive and that always appears on a clean screen at line one, you would enter the following:

```
PROMPT $e[2J$e[1;1f$p$g
```

To make the same prompt always appear on line 24 (screen remains uncleared), enter:

```
PROMPT $e[24;1f$p$g
```

If you were in the BIN directory on C drive, the prompt would appear on line 1 or 24 as:

```
C:\BIN >
```

Mode of Operation Codes

ANSI.SYS also allows you to set attributes that determine how your display operates.

Set Graphics Rendition Codes: The first set of attributes, the Set Graphics Rendition (SGR) codes, allow you to change foreground and background color on your color display. It also will allow characters to be bold, underlined (on IBM Monochrome Display), blinking, reversed video, or invisible.

The format for entering the SGR codes is:

```
ESC[#;...;#m
```

where the values of # parameter correspond to those shown in Figure 2.

# Value	Attribute
0	All attributes off (normal white on black)
1	Bold on (high intensity)
4	Underscore on (IBM Monochrome Display only)
5	Blink on
7	Reverse video on
8	Canceled on (invisible)
30	Black foreground
31	Red foreground
32	Green foreground
33	Yellow foreground
34	Blue foreground
35	Magenta foreground
36	Cyan foreground
37	White foreground
40	Black background
41	Red background
42	Green background
43	Yellow background
44	Blue background
45	Magenta background
46	Cyan background
47	White background

Figure 2. Set Graphics Rendition Parameters

For example,

```
ESC[44m
```

or

```
PROMPT $e[44m
```

would give you a blue background, and

```
ESC[44;33m
```

or

```
PROMPT $e[44;33m
```

would set screen attributes to display a blue background with yellow foreground.

Single words or phrases within text that will be written to the display can appear bold, blinking, reverse video, or in a different color. With an ASCII text editor, you can imbed the SGR codes within text.

Set Mode Codes: The second set of display attribute codes are the Set Mode (SM) codes. With these codes, you can set the screen width and type. The format for entering the SM codes is:

```
ESC[=#h
```

where the value of the # parameter corresponds to those shown in Figure 3.

# Value	Screen Attributes
0	40x25 black and white
1	40x25 color
2	80x25 black and white
3	80x25 color
4	320x200 color
5	320x200 black and white
6	640x200 black and white
7	Wrap at end of line (typing past end-of-line creates a new line)

Figure 3. Set Mode Parameters

For example:

ESC[= 1h

or

PROMPT \$e[= 1h

sets your display as a 40x25 color display.

Reset Mode Codes: The final set of display attribute codes are the Reset Mode (RM) codes. These codes are essentially the same as the SM codes, except that parameter 7 resets the end-of-line wrap (the characters past end-of-line are thrown away).

The format for entering RM codes is:

ESC[= #1

or

PROMPT \$e[= #1

where the value of the # parameter corresponds to those shown in Figure 3.

ANSI.SYS is a very versatile device driver. Once you begin using its capabilities, you will wonder how you ever got along without ANSI.SYS.

TopView Questions and Answers

(Part 1)

*Compiled by TopView Development Team
IBM Corporation*

The following questions and answers are intended to assist developers of IBM Personal Computer applications in evaluating TopView and the TopView Programmer's ToolKit.

Questions and answers fall into these categories:

- General
- Multitasking
- Windowing
- Data Exchange
- Program Information File
- TopView Programming Conventions
- Language Interface Support
- Mouse Support
- Device Driver Support
- Problem Determination



Part 1 covers the General and Multitasking categories; Part 2, coming next month, will cover the remaining categories.

General

Q1: What does TopView do?

A: TopView provides a multitasking and windowing operating environment for applications.

TopView features:

- Switching between co-resident programs.
- Concurrent execution of multiple tasks and programs.
- Moving, sizing, and scrolling an application's windows.
- Display of multiple windows at one time.
- Mouse support.
- Data exchange within and between applications.
- Ease-of-use features such as pop-up menus, Help windows, on-line tutorial, and access to commonly used DOS facilities.

- Support for text applications on both monochrome and color displays, and for graphics applications on a color display.

Many existing applications can run in the TopView environment, although they generally take advantage of only a subset of TopView facilities. At a minimum, existing applications permit switching between co-resident applications. However, all TopView facilities can be included in applications that are specifically designed to run with TopView (i.e., programmed using the TopView Application's Programming Interface).

Q2: Is TopView an operating system? If not, what operating system(s) and versions does TopView require?

A: TopView is not an operating system. It works in conjunction with DOS version 2.00 and higher to provide a multitasking and windowing operating environment. Note that under DOS 3.00 and 3.10, TopView supports only those functions that are available when running with DOS 2.00 or 2.10.

Q3: What applications will run with TopView now?

A: Many programs have been designated as "compatible with TopView" by IBM or the vendors that wrote them. These programs have been tested in the TopView environment by the vendors and can coexist with other programs running with TopView. However, not all TopView facilities may be available when using that program. For example, a program that writes directly to the video buffer cannot be windowed and cannot execute while in the background.

A list of applications that have been tested in the TopView environment can be found in the *TopView Application Guide for IBM Applications*, and in the *TopView Application Guide for Applications Available from Non-IBM Sources*.

Q4: What types of applications are best suited for the TopView environment?

A: The types of programs best suited for the TopView environment include:

- Programs written specifically to use the TopView I/O facilities.

- Programs that use DOS calls for video display rather than writing directly to video memory. Alternatively, programs that use the TopView INTerrupt I/O convention to allow TopView to provide the video buffer address and redraw facilities.
- Programs that do not excessively poll the keyboard.
- Programs that do not control hardware devices directly.
- Programs that are moderate in size or that use overlays so that more programs can be run under TopView at the same time.
- Programs that close files when the files are not in use.

Q5: How does TopView affect the performance of compatible applications?

A: TopView should have very little effect on the performance of most applications. However, the performance of an application in the TopView environment may vary depending on several factors that are not under TopView's control:

- The number of other applications running concurrently.
- The coding techniques used for each application (e.g., constant polling of a device versus waiting for an interrupt).
- The types of resources being used by each application (e.g., compute-bound, heavy I/O use).

Q6: Is there a size limitation for applications running with TopView?

A: There are practical limits determined by the amount of system memory, the size of DOS plus TopView, and the total size of all applications running concurrently.

TopView supports up to 640KB of RAM system memory. On an IBM PC with 256KB of system memory, approximately 80KB is available for the execution of application programs with DOS 2.00 and 2.10. With DOS 3.00 and 3.10, approximately 68KB is available for application programs.

Q7: What hardware is required to run TopView?

A: The minimum hardware required to run TopView is:

- An IBM Personal Computer AT, IBM Portable Personal Computer, IBM Personal Computer XT, or IBM Personal Computer.

- At least 256KB of RAM system memory.
- Either two double-sided diskette drives or one double-sided diskette drive and one fixed disk drive.
- An IBM Monochrome Display, IBM Color Display, or IBM Enhanced Color Display (running in emulation mode), and the appropriate adapter for your display.

Optional hardware includes:

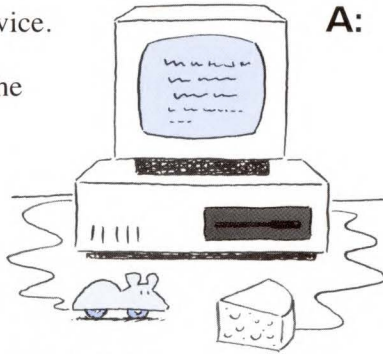
- Up to 640KB of RAM system memory (512KB or more is recommended).
- An IBM or compatible printer with the appropriate adapter.
- A mouse pointing device.

Q8: Does TopView run on the IBM PCjr?

A: No. TopView will not run on the IBM PCjr.

Q9: Does TopView run on the IBM 3270 PC?

A: Yes, TopView will run on the 3270 PC with Control Program version 1.20.



Q10: Does TopView provide an open Applications Programming Interface (API) so that any developer can write applications that take advantage of the TopView function?

A: Yes. The TopView Programmer's ToolKit documents the TopView API. The ToolKit provides the programming and language interface tools that the developer needs, and provides guidelines for the development of applications that can run in the TopView environment.

The TopView Programmer's ToolKit provides:

- Documentation of the TopView Applications Program Interface (API).
- The Window Design Aid for creating windows.
- Several utilities for handling windows built with the Window Design Aid and for building a Program Information File.
- Macros and routines for interfacing the IBM assembly language to TopView.
- A sample interface to IBM Pascal, illustrating how to write interfaces for higher-level languages.

- Two sample higher-level interfaces for the IBM assembly language and IBM Pascal.

Q11: What hardware is required to run the TopView Programmer's ToolKit utilities?

A: Hardware requirements for the TopView Programmer's ToolKit utilities are the same as the hardware requirements listed above for TopView itself, except that a mouse pointing device is highly recommended.

Q12: Does TopView support communications with other IBM PCs or with a host?

A: An application that runs with TopView could support communications with other IBM PCs or with a host.

It is not the intent of TopView to provide applications. TopView's primary function is to provide an environment that has multitasking and windowing facilities. Software developers can use these facilities to build programs that are more easily combined into integrated packages. TopView does include several small applications that serve as examples of programs that use multiple windows.

Q13: Does TopView support graphics applications?

A: Yes. Graphics applications run only in the foreground (interactively) and use the full screen. However, TopView does not require graphics, so users are not required to have a color monitor, graphics adapter or graphics software in order to run TopView.

Q14: Does TopView support BATch files?

A: No.

Q15: Who developed TopView and the TopView Programmer's ToolKit?

A: TopView and the TopView Programmer's ToolKit were designed and developed by the IBM Entry Systems Division in Boca Raton, Florida.

Q16: Why did IBM develop its own multitasking and windowing program?

A: TopView was developed to satisfy the requirements of those PC users who need greater productivity than is currently available in the PC environment.

Q17: Does TopView run on “compatible” personal computers?

A: TopView was designed, developed and tested to run on an IBM Personal Computer with DOS version 2.00 and higher. However, IBM has made no conscious design decisions that would preclude TopView from running on a compatible personal computer. TopView has not been tested on compatible personal computers.

Multitasking

Q18: What is an application? How is it defined to TopView?

A: An application is either:

- A program that currently runs in the DOS stand-alone mode using DOS version 2.00 or higher.
- A program that is aware of TopView and takes advantage of particular TopView features, but also can run as a stand-alone program under DOS.
- A program that is specifically written to run with TopView.

Q19: Why must an application be added to TopView’s Start-a-Program menu before it can run with TopView?

A: TopView must know the special operating characteristics of your program. This information is maintained in TopView’s Consolidated Program Information File.

Q20: What actually is installed when the application is added to TopView?

A: TopView uses the information from the application’s Program Information File or from the user to create a record in the Consolidated Program Information File, TV.PIF. This file contains the operating characteristics of all programs that can be started. Also, the name of the application is added to the “Start-a-Program” menu so that the user can start it.

Q21: Can users specify nicknames for applications?

A: Yes. When an application is added, the user can specify the name (up to 30 characters) by which the application is known to TopView.

Q22: Must each application be added after every IPL or power-on?

A: No. Once you add an application, TopView remembers the information it needs (unless the TopView Consolidated Program Information File TV.PIF is lost or damaged).

Q23: How many applications can be started at one time?

A: Theoretically, 255 windows can be managed at one time. An application uses one or more windows. However, there are practical limits—the size of the applications and the amount of system memory available. Only 23 applications can be displayed in the TopView switch menu at one time.

Q24: Is each application really independent of all other applications?

A: No. Applications running in the TopView environment are affected by each application’s use and access of resources that must be shared. If the recommended programming techniques are used to access these resources, TopView protects the application from other applications that also use the TopView facilities. However, all applications are vulnerable to other applications that directly access a resource.

Q25: If applications can run in the background (non-interactive) as well as in the foreground (interactive), how does TopView determine when to stop executing one application and start executing another one?

A: TopView uses a complex algorithm (time-slicing plus natural program breaks) to determine how much time an application receives and when switching is to occur. Applications are divided into two categories — those that are compute-bound and those that have break points (e.g., wait for an I/O operation to complete). A program waiting for an I/O event is moved to the top of the dispatch list when that event occurs, and therefore is dispatched more quickly than programs that are compute-bound.

Q26: Can the application developer or user determine or set the dispatching priority?

A: No.

Q27: Can an application developer protect a critical section of an application from being interrupted by other applications?

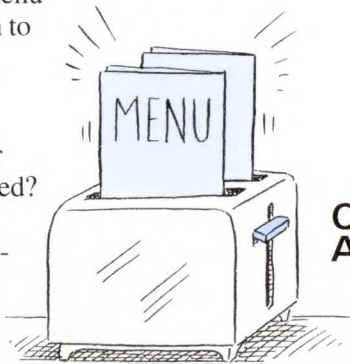
A: Yes. The TopView Applications Programming Interface (API) provides function calls that can define the beginning and end of a critical section of code.

Q28: Can the user find out what applications are currently started?

A: Yes. TopView provides a pop-up menu containing a list of all applications that are currently started. The menu can be used to switch to another application.

Q29: Can an application determine what other applications are started?

A: Indirectly. TopView does not give applications a direct facility for determining which applications are currently started. However, the application developer can determine if related applications are running by coding this function in a shared program.



Q30: Is there any way for an application to determine whether the printer or communication ports are being used by another application?

A: An open and subsequent write to either owned device will result in a critical error.

Q31: Can an application set any global flags that will affect the operating environment for other applications?

A: Yes. Refer to the TopView Programmer's ToolKit Reference book, Chapter 5, "Window Manager Command Bytes," for command bytes that apply to the system as a whole and not just to the application issuing the data stream.

Q32: If it becomes necessary to reboot, can TopView remember the program variables that have been set?

A: The basic TopView configuration is remembered (i.e., the items specified during Setup). The applications that were being used before rebooting are not remembered.

Q33: Is there a difference between a task and an application?

A: Yes. An application can have one or more tasks. A task is defined by creating a window and supplying a pointer to a portion of code (the task). If the pointer is 0 (zero), there is no task associated with the window. Each task is associated with a window, but the window may be 0 X 0 in size. Any window can be updated by any task that knows the handle address of the window.

Multiple windows can be associated with the same task. Each task is dispatched separately (i.e., each window with an associated task is dispatched separately).

Tasks are dispatched in a round-robin fashion, the order of which is not predictable or controllable by the application or the user.

Q34: How many tasks can an application have?

A: An application can have any number of tasks, subject to a TopView limit of 255 windows (a task must have one or more windows). In addition, the number of windows is constrained by the amount of system memory available.

Q35: Can tasks communicate with other tasks under an application?

A: Yes. TopView provides each task with a mailbox and a way to send and receive data. Applications also can use the shared program facility to communicate with other applications.

Q36: What data (application, TopView, DOS) is saved when TopView switches from one task to another?

A: Refer to the TopView Programmer's ToolKit Reference book, Chapter 8, "Task State Information Saved by TopView," for a list of items that are saved and restored on a task basis.

Q37: Does TopView allow use of the same object by different applications?

A: Each task can open and use separate objects from the same object class, but one task cannot use another task's specific object unless it can find out the object's handle. Object handles are not provided automatically. They would have to be communicated from one application to another via task mailboxes or a shared program data area.

Q38: Is the application that is receiving input from the keyboard or pointing device the only task that is executing?

A: No. TopView allows applications to execute concurrently, whether the application is interactive (receiving keyboard input) or not. However, TopView also allows the user to control which non-interactive applications are executing (by suspending the application). Also, there is a class of existing applications that cannot execute except when they are interactive (e.g., applications that write directly to the video buffer).

Q39: Does TopView support a wait/post mechanism? If so, what can an application wait for?

A: Yes. An application can wait for OBJECTQ, MAILBOX, KEYBOARD, POINTER, and TIMER.

Q40: Can an application wait for more than one event at a time?

A: Yes, via the OBJECTQ.

Q41: Are interrupt handlers interrupted by TopView?

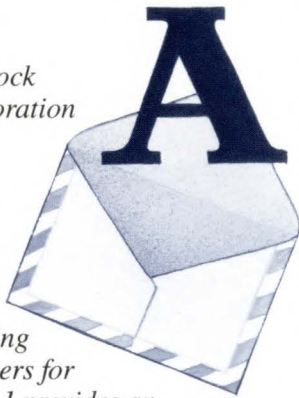
A: No. First-level interrupt handlers are not interrupted by TopView.

DOS Device Drivers

(Part 1)

John Warnock
IBM Corporation

Editor's note: This is the first part of an article about writing device drivers for DOS. Part 1 provides an overview of what device drivers do and how they work. Part 2, to be published next month, examines the functions that device drivers support and explains how to install and use device drivers.



The IBM Personal Computer Disk Operating System (DOS) uses programs called "device drivers" to communicate with the devices attached to your computer. DOS has built-in drivers to support the standard devices that attach to members of the IBM Personal Computer family. You don't have to do anything special to use these drivers—when one of your programs reads or writes to device "A:" or "LPT1:", DOS automatically uses the correct device driver.

However, if you want to attach a special or unusual device to your computer or change how an existing device performs, you may need to supply your own device driver—a program that translates DOS's read and write requests into the specific commands that your device understands.

This article describes how to create device drivers, how to make them part of your system, and how to communicate with them using commands and parameters available in DOS.

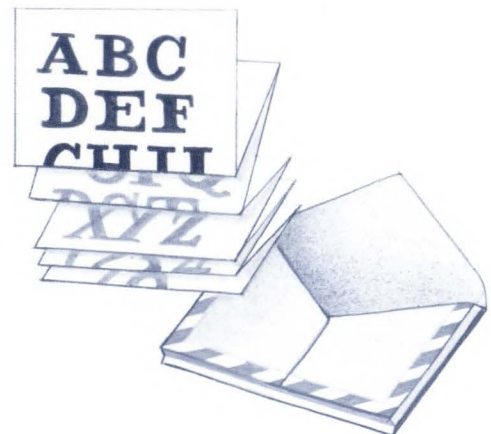
Functions of Device Drivers

Device drivers serve two basic functions. They act as translators, converting the pulses and signals between the computer and the device. They also act as emulators, making an unfamiliar piece of hardware acceptable to the computer.

Translators modify input and output to satisfy the requirements of both the system and the attached device. Examples of translators are Dvorak keyboard drivers, file encryption drivers, foreign language keyboards, and graphic display drivers. A bisynchronous device driver that converts ASCII character codes to EBCDIC characters would be a translator.

Emulators enable the computer to view a device as either a character device or a block device, the only two kinds of devices the computer recognizes. Emulation takes a strange, unsupported device and makes it appear to the computer as a familiar, supported device. A device driver that takes memory and makes it appear to the computer as a diskette is an example of an emulator.

Character devices, such as the keyboard (CON) and printer (LPT1), send and receive one character at a time. A character device driver refers to a specific device name in its program header, so it applies only to that device. Two examples of character device drivers are a parallel printer emulator that uses the Asynchronous Communications



DOS 3.10 Reference Chart

Compiled by John Warnock, IBM Corporation

This reference chart may be removed and used for future reference. It includes the following information:

- DOS 3.10 Commands
- DOS 3.10 Special Characters and Extensions
- DOS 3.10 Special Function Key Assignments and Device Names
- ANSI.SYS ESCape Codes

DOS 3.10 Reference Chart

DOS 3.10 Commands

DOS COMMAND	OPTIONS	TYPE	FUNCTION
ANSI.SYS		Config	Keyboard device driver
ASSIGN	x=y	External	Assign drive name to another drive
ATTRIB	+R -R	External	Check or change read-only file attribute
AUTOEXEC.BAT		File	BATch file support on system start-up
BACKUP	/S/M/A/D	External	Backup disk(ette) files to disk(ette)
BASIC	/F/S/C/M /D	External	BASIC language interpreter
BASICA	/F/S/C/M /D	External	Advanced BASIC language interpreter
BREAK	ON OFF	Internal	Enables/disables check for Ctrl-Break
BREAK=	ON OFF	Config	Enables/disables check for Ctrl-Break
BUFFERS=		Config	Allocate disk buffers
CD		Internal	Change directory
CHDIR		Internal	Change directory
CHKDSK	/F/V	External	Check diskette or fixed disk
CLS		Internal	Clear screen
COMMAND	/P/C	External	Starts second DOS command processor
COMP		External	Compare files
COMSPEC		Set	Specify path to reload command processor
CONFIG.SYS		File	Configuration file Commands
COPY	/A/B/V +	Internal	Copy files
COUNTRY=		Config	Set date & time format, currency symbols
CTTY		Internal	Substitute I/O for screen and keyboard
DATE		Internal	Set date
DEBUG		External	Load, alter, display/execute files
DEL		Internal	Deletes specified file(s)
DEVICE=		Config	Specify device driver program
DIR	/P/W	Internal	List directory
DISKCOMP	/1/8	External	Compare diskettes
DISKCOPY	/1	External	Copy diskette contents
ECHO	ON OFF	BATch	Controls display of DOS commands
EDLIN		External	Line editor
ERASE		Internal	Erases specified file(s)
EXE2BIN		External	Converts .EXE files to .COM format
EXIT		Internal	Terminates second DOS session
FCBS=		Config	Set no. of available file control blocks
FDISK		External	Partition a fixed disk
FILES=		Config	Specify maximum number of open files
FIND	/V/C/N	Filter	Search for string in file
FOR IN DO		BATch	Iterative execution of DOS commands
FORMAT	/S/1/8/V /B/4	External	Initialize a diskette or fixed disk
GOTO		BATch	Branches to a label
GRAFTABL		External	Load graphics character table

DOS 3.10 Special Characters and Extensions

.BAT	File	File with batch commands
.COM	File	File with memory image of program
.EXE	File	File with program
%n	BATCH	Replaceable parameter (n = 1-9)
:	BATch	Label designator
	Piping	Redirects input/output to filter
<	Redirector	Input source
>	Redirector	Output destination

DOS 3.10 Keys and Devices

FUNCTION KEY	FUNCTION
F1	Types previous command, one character at a time
F2	Types command up to designated character
F3	Types all of previous command
F4	Deletes command to designated character
F5	Restarts command entry, saving current command
F6	^Z Generates end of file
Del	Skips over character in stored command
Esc	Restarts command entry
Ins	Adds characters to stored command
DEVICE	DESCRIPTION
A:, B:, etc.	Block devices—input or output
CAS1	Cassette tape rcdr—input or output
CLOCK\$	“Real Time Clock” card—input or output
COM1	1st Asynch adapter—input or output
COM2	2nd Asynch adapter—input or output
CON	Console keyboard/screen—input or output
KYBD	Keyboard—input only
LPT1	1st parallel printer device—output only
LPT2	2nd parallel printer device—output or random
LPT3	3rd parallel printer device—output or random
NUL	Dummy (nonexistent) device—input or output
PRN	1st parallel printer—output only
SCRN	Screen—output only

ANSI.SYS ESCape Codes

CURSOR CONTROL SEQUENCES	
COMMAND	DESCRIPTION
ESC[#;#H	Specify cursor position by row and column number. Default 1,1
ESC[#A	Move cursor up # rows. Default 1
ESC[#B	Move cursor down # rows. Default 1
ESC[#C	Move cursor forward # columns. Default 1
ESC[#D	Move cursor backward # columns. Default 1
ESC[##;#f	Specify cursor position by line and column number. Default 1,1
ESC[##;#R	Report cursor position (line; column) through standard input device
ESC[6n	Console driver outputs a cursor position report when device status report received.
ESC[s	Save the current cursor position.
ESC[u	Restore cursor position saved with ESC[s.
ESC[2J	Erase screen, cursor goes to home position.
ESC[K	Erase from cursor position to end of line.

KEYBOARD KEY REASSIGNMENT	
ESC[##;...#p or ESC["string"p or ESC[##;"string"p	First parameter is an ASCII code to be mapped. Remaining parameters are redefinitions that

GRAFTABL	/R/B	External	Load graphics character table
GRAPHICS		External	Graphics screen dump MODIFIED for PrtSc
IF [NOT]		BATch	Conditionally executes DOS command
JOIN	/D	Internal	Joins two or more directories
KEYBxx		External	Load keyboard support program
LABEL		External	Change volume ID of disk or diskette
LASTDRIVE =		Config	Set maximum no. of drives to access
LINK		External	Link-edit a compiled program for executable output
MD		Internal	Create a subdirectory
MKDIR		Internal	Create a subdirectory
MODE		External	Set display or printer options
MORE		Filter	Pause after displaying screenful
PATH		Internal	Specify directory paths for programs and commands
PAUSE		BATch	Prompt with a wait for a keystroke
PRINT	/D/B/U/M /S/Q/C /T/P	External	Print files in background
PROMPT		Internal	Change DOS prompt
RD		Internal	Remove directory
RECOVER		External	Recover file
REM		BATch	Remark
REN		Internal	Rename a file
RENAME		Internal	Rename a file
RESTORE	/S/P	External	Restore files from disk(ette) to disk(ette)
RMDIR		Internal	Remove directory
SELECT		External	Select country support
SET		Internal	Set or display environment parameters
SHARE	/F/L	External	Install file sharing support
SHELL =		Config	Designates other command processor versus COMMAND.COM
SHIFT		BATch	Allows command line to exceed 10 variables
SOFT	/R/ + n	Filter	Sort data
SUBST	/D	Internal	Substitutes path string for file
SYS		External	Transfer DOS to diskette or fixed disk
TIME		Internal	Set time
TREE	/F	External	Display directory paths
TYPE		Internal	Display file contents
VDISK.SYS		Config	Virtual RAM disk device driver
VER		Internal	Display DOS version
VERIFY	ON OFF	Internal	Verify data properly written to disk(ette)
VOL		Internal	Display disk(ette) label

or ESC["string"p or ESC[;"string";#; #;"string";#p or any other combination of strings and decimal numbers	Remaining parameters are redefinitions that are executed when mapped key is intercepted. If the first parameter is a 0 (null) then the first and second parameters make up an extended ASCII code that is redefined by the succeeding parameters.
---	--

MODE OF OPERATION		
Set Graphics Rendition (SGR)		
ESC[#;...;#m	Set character attribute, #, according to the following table:	
	# Value	Attribute
	0	All attributes off (normal white on black)
	1	Bold on (high intensity)
	4	Underscore on (IBM Monochrome Display only)
	5	Blink on
	7	Reverse video on
	8	Canceled on (invisible)
	30	Black foreground
	31	Red foreground
	32	Green foreground
	33	Yellow foreground
	34	Blue foreground
	35	Magenta foreground
	36	Cyan foreground
	37	White foreground
	40	Black background
	41	Red background
	42	Green background
	43	Yellow background
	44	Blue background
	45	Magenta background
	46	Cyan background
	47	White background
Set Mode (SM)		
ESC[=#h or ESC[=h or ESC[=0h or ESC?7h	Set screen type and width, #, according to the following table	
	# Value	Screen attributes
	0	40x25 black and white
	1	40x25 color
	2	80x25 black and white
	3	80x25 color
	4	320x200 color
	5	320x200 black and white
	6	640x200 black and white
	7	Wrap at end of line (Typing past EOL creates a new line)
Reset Mode (RM)		
ESC[=#1 or ESC[=1 or ESC[=01 or ESC[?71	Same as Set Mode (SM) except parameter 7 resets wrap at end-of-line mode (characters past end-of-line are thrown away.)	

Adapter and a serial printer, and a console emulator that uses the Asynchronous Communications Adapter and a remote terminal to control the computer.

Block devices, such as diskette drives A and B, send and receive several characters at a time. Block device drivers do not refer to a specific device name, but are generic in nature. A block device driver defines the number of devices it handles, and DOS assigns device names accordingly. One disk device driver, for example, could handle drives A, B, C and D.

To make use of a virtual disk in system memory, you need a block device driver. If a system uses a virtual disk device driver that handles three devices, DOS will assign the next available letters to these new drives. For example, on a system with two diskette drives but no fixed disk drives, the three virtual drives would be C, D, and E.

DOS 3.00 and DOS 3.10 accept up to 26 block devices; DOS 2.00 accepts 16; and DOS 2.10 accepts 63. Once DOS 2.10 assigns the 26 letters of the alphabet, it resorts to assigning characters by collating sequence. You can expect devices named "!", "@", "&", and so on.

DOS 2.10 VDSK

You can see a sample virtual disk device driver called VDSK on pages 3-27 through 3-34 of the DOS 2.10 Technical Reference manual. VDSK defines a single 180KB virtual disk. The size of the disk and sector size are fixed. A more advanced version of the program can be found on the supplemental program disk of DOS 3.00 and 3.10 which we will discuss later.

OFFSET	LENGTH	DESCRIPTION
+00	3 bytes	NEAR JUMP to boot code
+03	8 bytes	Original Equipment Manufacturer name and version
+11	2 bytes	Number of bytes per sector
+13	1 byte	Sectors per allocation unit (must be a power of 2)
+14	2 bytes	Reserved sectors (starting at logical sector zero)
+16	1 byte	Number of FATs (File Allocation Tables)
+17	2 bytes	Number of root directory entries (maximum allowed)
+19	2 bytes	Number of sectors in logical image (total sectors in media, including boot and directory sectors)
+21	1 byte	Media descriptor byte
+22	2 bytes	Number of sectors occupied by a single FAT
+24	2 bytes	Number of sectors per track
+26	2 bytes	Number of heads
+28	2 bytes	Number of hidden sectors

Figure 1. BIOS Parameter Block

The DOS 2.10 VDSK block device driver is an assembly language program for a device called VDSK. This device driver shows how the header references are organized, and because it is a block device, it contains a character device name which is ignored. (In fact, DOS sets the first byte to the number of devices handled by the driver.) The VDSK block device driver has a function table, device strategy routine, device interrupt routine, function subroutines and common exit code. It also contains a BIOS Parameter Block (shown in Figure 1) and virtual disk information, neither of which is used for character devices).

The BIOS Parameter Block contains media format information for devices with removable media. For example, if you remove a single-sided diskette and insert a double-sided diskette, your driver

should update the BIOS parameter block to reflect these changes.

In the BIOS Parameter Block, the media descriptor byte accommodates several media types. This byte is critical when you write to a removable media device whose media formats can change. DOS requests the driver to perform a media check. If the driver responds that the media has been changed, DOS requests the driver to build a new BIOS Parameter Block. Information for the new block is taken from the boot sector of the media and includes a media descriptor byte.

The four high-order bits in the media descriptor byte should always equal hex F. Only the low three bits change.

Common values for the media descriptor byte are shown in Figure 2. (Figure 2 contains information for all versions of DOS.)

DISK TYPE	# OF SIDES	SECTORS /TRACK	DESCRIPTOR VALUE	COMMENTS
5 1/4 "	1	9	FCH	Single-sided DOS 2.00 +
5 1/4 "	2	9	FDH	Double-sided DOS 2.00 +
5 1/4 "	1	8	FEH	Single-sided DOS 1.00 +
5 1/4 "	2	8	FFH	Double-sided DOS 1.00 +
Fixed			F8H	Non-removable media
5 1/4 "	2	15	F9H	High-capacity DOS 3.00 +
8 "	1	26	FEH	IBM 3740 format: single-sided, single-density
8 "	2	26	FDH	IBM 3740 format: double-sided, single-density
8 "	2	8	FEH	Double-sided, double-density, 1024 byte sectors

Figure 2. Media Descriptor Byte

Figure 2 shows two instances where different media use the same media descriptor byte (FEH, FDH). This is not an error. In some instances, media of different physical sizes share common characteristics. Because DOS communicates directly with only the device driver, the device driver must concern itself with physical size. Also, it is unlikely that you would write one driver for two different media.

In the case of the two 8-inch diskettes sharing the FEH descriptor byte, the device driver should test for media. If the device driver gets an error when trying to read a single density address mark, the media should be considered double density. An error on reading from the second read/write head would indicate single-sided media.

DOS 3.00/3.10 VDISK

The DOS 3.00 and 3.10 program diskettes contain VDISK, a much more extensive virtual disk device driver. VDISK lets you set the disk size, sector size, the number of directory entries, and the use of extended memory (starting address at 1MB). You can access this driver through a DEVICE state-

ment in a CONFIG. SYS file. Each additional copy of the VDISK driver you call uses 800 bytes of overhead.

The VDISK device driver contains not only the common header for device drivers but also individual headers for each type of request (read, write, etc.). With many explanatory comments, it shows how to set up a table to direct these requests to the appropriate routine in the driver.

(An assembly listing of the VDISK program is available on the DOS 3.00 and DOS 3.10 supplemental programs diskette in the file VDISK.LST. To print it, use the DOS PRINT command; you will get 59 pages in compressed print mode.)

The VDISK driver in DOS 3.00 and 3.10 contains headers and routines similar to the VDSK driver in DOS 2.10. Through its code and comments, VDISK gives substantial information about how disk and diskette drives work. The VDISK device driver also provides routines for using the extended memory of the IBM Personal Computer AT and the new instructions and error codes of DOS 3.00 and 3.10.

Designing a Device Driver

A device driver is a program with either an origin of zero (ORG 0) or no origin at all. The program begins with the device header, as shown in Figure 3.

NAME	LENGTH	NOTES
Pointer to next device	4 bytes	Usually -1 (FFFFFFFF) for one device driver in the file
Attribute word	2 bytes	A 1 in each bit enables: Bit 0 = standard input Bit 1 = standard output Bit 2 = nul device Bit 3 = clock device Bit 11 = open/close removable media supported (DOS 3.00 and 3.10) Bit 13 = non-IBM format Bit 14 = IO-CTL used Bit 15 = character device (All other bits must be off)
Device strategy routine pointer	2 bytes	
Device interrupt routine pointer	2 bytes	
Name of character device or number of block devices	8 bytes 1 byte	Defined by device driver Returned by DOS

Figure 3. Device Header

The device header is followed by data and functional code. The code requires both a device strategy routine and a device interrupt routine because DOS approaches the device driver twice to handle one request. The device strategy routine receives the first request from DOS and saves a pointer to the request header. The strategy routine enqueues the request, complete with parameters for the interrupt routine. When the strategy routine returns to DOS, DOS immediately places a second request to the interrupt routine which actually performs the request and returns its status to DOS.

The device interrupt routine should set the ES and BX registers with these instructions before returning control to DOS:

```
MOV    ES,CS:RH_SEG
MOV    BX,CS:RH_OFF
```

Any functions not used by your device driver should return the status error code for an unknown command:

```
STATUS DONE, ERROR,
0003H
JMP    EXIT
```

Your device driver must support the command codes shown in Figure 4 in order to work properly with DOS.

The way you implement these command functions will be unique to the device and the device driver. The IOCTL functions are particularly useful if your device will accept them. They allow control information to be exchanged with a device without actually doing a read or write. This is useful for setting baud rates, type fonts and other control information.

CMD	NAME	CHARACTER	BLOCK
0	Init	X	X
1	Media Check		X
2	Build BIOS Parameter Block		X
3	IOCTL Input	X	X
4	Input	X	X
5	Nondestructive Input No Wait	X	
6	Input Status	X	
7	Input Flush	X	
8	Output Write	X	X
9	Output Write with Verify	X	X
10	Output Status	X	
11	Output Flush	X	
12	IOCTL Output	X	X
13	Device Open (DOS 3.00, 3.10)	X	X
14	Device Close (DOS 3.00, 3.10)	X	X
15	Removable Media (DOS 3.00, 3.10)		X

Figure 4. Device Driver Command Codes

OFFSET	LENGTH	DESCRIPTION
+00	1 byte	Length of header + data
+01	1 byte	Unit code (block devices only)
+02	1 byte	Function code (command code)
+03	2 bytes	Status code (set by device)
+05	8 bytes	Reserved for DOS
+13	1 byte +	Function data

Figure 5. Request Header

BITS	DESCRIPTION
15	Error bit set when low 8 bits carry error
14-10	Reserved
9	Busy bit for Status and Removable Media
8	Done bit—set to 1 by driver when task is completed
7-0	Error code—valid only when bit 15 is on <ul style="list-style-type: none"> 00 = Write protect violation 01 = Unknown unit 02 = Device not ready 03 = Unknown command 04 = CRC error 05 = Bad drive request structure length 06 = Seek error 07 = Unknown media 08 = Sector not found 09 = Printer out of paper 0A = Write fault 0B = Read fault 0C = General failure 0D = Reserved 0E = Reserved 0F = Invalid disk change

Figure 6. Status Code Word Bit Configuration

Three Basic Operations

Your device driver program should contain command functions that perform three basic operations:

- Read the request header and data in the device strategy routine.

- Act upon the request.
- Return the requested information to DOS in the device interrupt routine.

The request header is a particular kind of function header. It is shown in Figure 5.

The fourth field of each request header is a two-byte status code.

Your device driver must set that status when it completes its task. The bit configuration of the status code word is shown in Figure 6.

Customize Your DOS Prompt

Jerry Schneider

Capital PC User Group

Are you tired of seeing the DOS prompt, `A >`, glaring at you from your display screen? When you use subdirectories, do you ever forget what subdirectory you are currently working in? Would you like the time displayed with the prompt? These problems can be easily remedied by using the `DOS PROMPT` command.

The `PROMPT` command lets you specify what will appear on your screen as the prompt. You can specify text as your prompt, such as Good Morning, or, by using special DOS predefined meta-strings, you can customize prompts to display the date, time, subdirectory, or a variety of other features.

The DOS predefined meta-strings are in the format `%x`, where `x` can be one of the following characters:

Character	Display at Prompt
\$	The \$ character
t	The time
d	The date
p	The current directory of the default drive.
v	The DOS version number
n	The default drive letter
g	The > character
l	The < character
b	The : character
l	The = character
h	A backspace; the previous character of the prompt is erased
e	The ESCape character
—	The CR LF sequence (go to beginning of a new line on the display screen).

For example, if you wanted to display a message in addition to the usual default drive letter and familiar `>` character, you could enter:

```
PROMPT Good Morning, Jerry %n$g
```

and your prompt becomes

```
Good Morning, Jerry A >
```

If you want your prompt to display the date and time, as well as the default drive and `>` character, enter:

```
PROMPT %d %t %n$g
```

and the following prompt is displayed:

```
Sat 9-28-1985 15:38:49.16 A >
```

The following example sets the DOS prompt to display the current directory of the default drive plus the `>` character:

```
PROMPT %p$g
```

If the default drive is `A` and the current directory of drive `A` is `\JERRY\FW`, the DOS prompt would display:

```
A:\JERRY\FW >
```

If you wanted the prompt to display just the time (without the seconds) on one line, and the drive letter, current directory of the default drive, and the `= >` characters on a second line, you would enter:

```
PROMPT %t$hh$hh$hh$hh$hh$_%n$g$g
```

which would display the following two-line prompt:

```
15:38
A:\JERRY\FW = >
```

After you have experimented with the `PROMPT` command and decided how you want your prompt to look, enter the `PROMPT` command (with parameters) as part of your `AUTOEXEC.BAT` file, where it will automatically be executed whenever you start up your system.

DOS Filter Commands

John Warnock
IBM Corporation

DOS versions 2.00 and higher contain several filter commands and operations that you can use to customize file listings. Filters are programs that take input from files, programs or other devices and modify the information before sending it on to another program or device. Filters usually require no user prompts. You can write your own filters, even in BASIC.

By using filtering processes, you can:

- Use all or part of the information available in a file.
- Modify the information you have selected.
- Redirect the modified information to an output file, output device, or additional filter.

The DOS filter commands are FIND, MORE and SORT. These are all external commands, so they must be in the current disk directory or else have a path specified when they are called. The filter operations use Piping (|) and Input/Output Redirection (< and >).

Piping and I/O Redirection

The Piping and Input/Output (I/O) Redirection operations specify where information comes from and where it goes to.

The Piping (|) operation should be entered between two DOS commands. It specifies that the output of the first command is the input for the second command. For example:

```
DIR | SORT
```

tells DOS to send the output of the DIRectory command as input to the SORT command.

If the SORT input does not come from another DOS command, or if the SORT output goes to a file or device, then an I/O Redirection operation is necessary. For example, to SORT the file IN to the file OUT, the syntax is:

```
SORT <IN >OUT
```

The character "<" precedes the input file or device, and the character ">" precedes the output file or device.

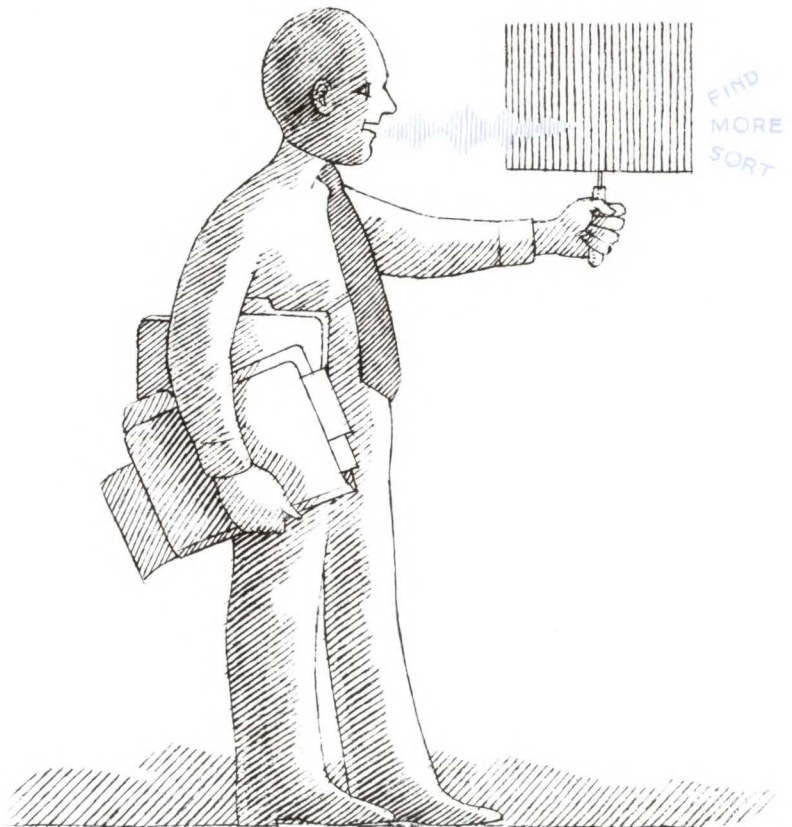
I/O Redirection lets you redirect normal DOS or application output to files and communication lines as well. The command:

```
DIR > LISTING
```

puts a directory listing into a file called LISTING. The file can then be printed with the DOS PRINT command or used by a word processing program that accepts ASCII text files. The instruction:

```
DIR > COM1:
```

redirects the directory listing to a serial printer attached to your communications adapter card.



Filter Commands

FIND command: This command locates lines that contain a specified character string or excludes lines that contain the specified character string. The format is:

```
FIND [/V][/C][/N] "string" [d:][path]
      [filename[.ext] ...]
```

where

/V displays lines that do not contain the string

/C displays a count of how many lines contain the string

/N displays the relative line number where the match occurred

"string" is the phrase or text you are trying to match, enclosed in double quotes

[d:][path][filename[.ext] ...] lets you specify consecutive input files to search with standard input. Piping omits the file.

For example, look for the string "to be" in the file TEXT1.TXT, type:

```
FIND "to be" TEXT1.TXT
```

To extend the search to include files TEXT2.TXT and TEXT3.TXT, type:

```
FIND "to be" TEXT1.TXT
      TEXT2.TXT TEXT3.TXT
```

To find all lines not containing the string "to be", and to display the relative line number of all lines found not containing "to be", type:

```
FIND /V /N "to be" TEXT1.TXT
      TEXT2.TXT TEXT3.TXT
```

MORE command: This command is similar to the TYPE command, but MORE allows you to read the information in the file one screenful at a time. When the screen fills, MORE stops scrolling and issues the message:

— More —

When you finish reading the information on the screen, you press any key to continue.

For example, to display the file ASM.LST so that you can read each screen, type:

```
MORE < ASM.LST
```

The MORE command requires an I/O Redirection or Piping operation. To use the MORE command with the TREE command, the DOS manual suggests typing:

```
TREE | MORE
```

SORT command: This command, as its name implies, reorders information before writing it back out. Its format is:

```
SORT [/R][/+n]
```

where

/R designates sorting information in reverse order

/+n designates sorting information by the characters starting in column n

Like the MORE command, SORT relies on I/O Redirection and Piping operations to tell it where the information can be found and where it should go.

Using Filter Commands

You can combine several filter commands and DOS commands. The following examples illustrate the filter commands and operations combined with the DIR command.

To alphabetically sort a directory listing and have the sorted list pause when the screen fills up, type,

```
DIR | SORT | MORE
```

You may want to keep an alphabetical directory list with each diskette in order to find files more easily. To send an alphabetically sorted directory listing to the printer, type:

```
DIR | SORT >|pt1:
```

To display an alphabetical listing of subdirectories only, type:

```
DIR | SORT | FIND "<DIR>"
```

or

```
DIR | FIND "<DIR>" | SORT
```

To display a listing of files sorted by extension and excluding subdirectories, type:

```
DIR | SORT /+10 | FIND /V "<DIR>"
```

or

```
DIR | FIND /V "<DIR>" | SORT /+10
```

The /V tells FIND to omit lines containing <DIR>; the /+10 tells SORT to begin sorting according to the character found in column 10, the first position of the extension.

To list files in the subdirectory D:\SAMPLE from largest to smallest, type:

```
DIR D:\SAMPLE | SORT /R /+14
```

The /+14 is the starting position of the file size, and /R tells SORT to list the results in reverse (descending) order.

To get a printout by date of all the files with extension .COM, you can sort by month or year, but not both. To sort by month, the SORT command should begin sorting at column 24:

```
DIR *.COM | SORT /+24
```

To sort by year, the sort should begin at column 30. You can use the output director > to send the output to the printer. Type:

```
DIR *.COM | SORT /+30 >LPT1:
```

Filters also can be used outside of DOS. BASIC 3.00 includes a new command called SHELL, which lets you perform DOS commands from within BASIC. When combined with the SORT command, the SHELL command lets you sort and display a file you create in BASIC. The following is a sample program that performs this function:

```
10 REM SAVE "SAMPSORT.BAS"
20 DEFINT A-Z
30 CLS
```

The following section creates a file called ONE, and prompts for input. The length of 25 is arbitrary.

```
40 OPEN "ONE" FOR OUTPUT AS #1
   LEN=25
50 PRINT "Enter the items you want to sort"
```

The next section increases the item count by 1 and displays the new count, waiting for input. If there is some input, the section prints the input in file ONE; otherwise it skips to the next section.

```
60 I=I+1
70 PRINT STR$(I);
80 LINE INPUT ". ";N$
90 IF N$ < > "" THEN PRINT #1,N$ ELSE
   GOTO 110
100 GOTO 60
```

This section clears the screen, tells you it is calling SORT, and closes the ONE file so SORT can use it.

```
110 CLS
120 PRINT "Entry ended. Calling SORT."
130 CLOSE #1
```

The SHELL command calls for a SORT from file ONE to file TWO.

```
140 SHELL "SORT <ONE> >TWO"
```

The following section tells you the SORT is finished and readies the two files for reading.

```
150 CLS : I=0
160 PRINT "Sort Completed. Results:"
170 OPEN "ONE" FOR INPUT AS
   #1 LEN=25
180 OPEN "TWO" FOR INPUT AS
   #2 LEN=25
```

This section checks if the first file is not empty, reads both files, prints the results and repeats the cycle.

```
190 WHILE NOT EOF(1)
200 INPUT #1,A$
210 INPUT #2,B$ : I=I+1
220 PRINT STR$(I);". ";TAB(7);A$;
   TAB(35);B$
230 WEND
```

This section closes the files and ends the program.

```
240 CLOSE #1, #2
250 END
```


Watch Where You Copy To!

Michael Engelberg
IBM Corporation

If you use a fixed-disk-based Personal Computer and have divided your fixed disk into several subdirectories, you probably move files across directories by first copying files from one subdirectory to another and then erasing the files from the original subdirectory.

In the DOS COPY command, you specify the names of both the "from" and "to" subdirectories. If you misspell the name of the "to" subdirectory, and no other subdirectory has the misspelled name, the file will be copied somewhere else and will be given the name you misspelled.

For example, if you want to copy a file TRY.IT from subdirectory ONCE to subdirectory TWICE, you would normally type:

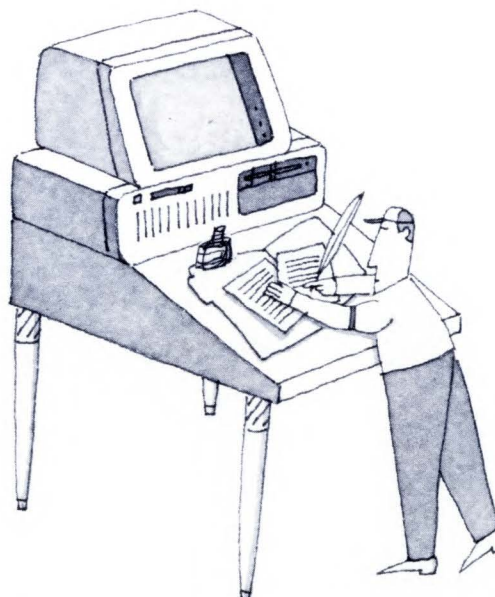
```
COPY \ONCE\TRY.IT \TWICE
```

Suppose you made a mistake typing in the command and you actually typed:

```
COPY \ONCE\TRY.IT \TRICE
```

If you have no subdirectory named TRICE, DOS still returns the message "1 file(s) copied", even though the file has not been copied into the subdirectory you intended. DOS first searched for a subdirectory named TRICE. Not finding a TRICE subdirectory, DOS treated \TRICE as the path to the root directory (\) followed by a new file name, TRICE. DOS then copied file TRY.IT from subdirectory ONCE into the root directory and gave the copied file the new name TRICE. Issuing the DIR command from the root directory would verify that the TRICE file has been added there.

Now suppose your final step is to erase the file TRY.IT from subdirectory ONCE. After you erase it, you change to subdirectory TWICE, look for file TRY.IT, and find it isn't there. At this point it isn't in subdirectory ONCE, either.



But all is not lost. If you saw the message "1 File(s) Copied", DOS has made a copy of TRY.IT somewhere. The copied file may be a bit difficult to find because you don't know its name (unless you remember your misspelling), but eventually you will find it. Usually it will be in the directory level that is immediately above the level you are in.

If ONCE and TWICE are second-level subdirectories under first-level subdirectory HOWOFTEN, and you incorrectly enter the command:

```
COPY \HOWOFTEN\ONCE\TRY.IT \
HOWOFTEN\TWICE
```

as

```
COPY \HOWOFTEN\ONCE\TRY.IT \
HOWOFTEN\TRICE
```

the COPY command will copy file TRY.IT from the second-level subdirectory ONCE into the first-level subdirectory HOWOFTEN, and will give the copied file the name TRICE. To locate the misspelled file name, change to the subdirectory HOWOFTEN and issue a DIR command.

Suggestion: After you copy a file from one subdirectory to another, use the DIR command to verify that the copied file exists in the target subdirectory before you erase it from the original subdirectory.

Once you have defined your ASCII graphics characters, it will be much easier to control what appears on the screen, because you have defined the screen as a collection of characters rather than dots. You now have to control only 2,000 characters (consisting of 80 columns per row and 25 rows) instead of 64,000 dots (200 x 320).

Assembly Language Program Defines Our Character

Following is the explanation of how to construct an assembly language subroutine that defines ASCII character number 128, the first of the upper 128 characters. This subroutine first defines the character using the “man” pattern shown above, and then it resets the upper character set pointer to point to the new character.

The subroutine shown below also provides an example of the syntax you will need in order to link the subroutine with a BASIC program.

Note: IBM has written and tested the programs contained in this article. However, IBM does not guarantee that the programs contain no errors.

The first two lines of code define the segment. The 'CODE' is necessary for adding this subroutine to BASIC's CODE segment.

```
CODE      SEGMENT 'CODE'
          ASSUME  CS:CODE
```

The next two lines define the procedure that is called from the compiled BASIC program.

```
          PUBLIC  LOADER
LOADER    PROC    FAR
```

The next statement jumps over the actual character definition which is not executable code.

```
          JMP     OVERDEF
```

Next is the data definition statement that defines the new character.

```
          DEF     DB      7EH,81H,42H,
                        3CH,24H,24H,
                        24H,0C3H
```

The next section resets the pointer to the upper 128 ASCII characters so that it points to our new character set. The pointer is located at segment 0, offset 7CH. We will first reset the offset pointer, then the segment pointer.

The following three lines set up the DS register to point to our data.

```
OVERDEF:  PUSH    DS
          MOV     AX,CS
          MOV     DS,AX
```

To reset the offset pointer, we move 7CH into BX and 0 into ES.

```
          MOV     BX,7CH
          MOV     AX,0
          MOV     ES,AX
```

Into AX we place the offset to our new characters.

```
          MOV     AX,OFFSET DEF
```

Now we can actually set the offset value.

```
          MOV     ES:[BX],AX
```

The segment pointer is located at the next word in memory (7E hex). The value we need is found in the current DS register.

```
          MOV     BX,7EH
          MOV     AX,DS
          MOV     ES:[BX],AX
```

We can now finish the subroutine.

```
          POP     DS
          RET
LOADER    ENDP
CODE      ENDS
          END
```

After you link this routine with a compiled BASIC program, you can call it with the following statement:

```
          CALL    LOADER
```

In your BASIC program you can now make your “man” character appear on the screen simply by using the statement:

```
          PRINT   CHR$(128);
```

and, after he moves, you can erase him using the statement:

```
          PRINT   CHR$(32);
```

which replaces the man with a blank character.

BASIC Program Animates Our Character

The following BASIC program gives an example of how to move the character we have defined above. The first line defines any variable that starts with a letter to be an integer.

```
10 DEFINT A-Z
```

Lines 20 and 30 set up the two variables that we will use to control our character's position. X is the current row and Y is the current column.

```
20 X = 1
30 Y = 1
```

Next we call the assembly language routine **LOADER**:

```
40 CALL LOADER
```

In line 50 we set our screen mode to 320 by 200 graphics:

```
50 SCREEN 1,0,0
```

The following code (the heart of the program) executes every time the character is moved. The **LOCATE** statement moves the cursor to the position at which we will write the character. The **PRINT** statement then prints the character that we defined in the assembly language subroutine.

```
100 LOCATE X,Y
110 PRINT CHR$(128);
```

This program assumes you will move the character using the keyboard's arrow keys. Lines 120 and 130 continually poll the keyboard to see if a key is pressed. Once it is pressed, the program checks to see if the key pressed was the Escape key, which signals the end of the session.

```
120 KEYED$=INKEY$
130 IF KEYED$="" THEN GOTO 120
135 IF KEYED$=CHR$(27) THEN END
```

If you press an arrow key, two characters are returned, because the arrow keys produce extended ASCII codes. (A list of these codes appears in the Appendix of the BASIC manual.) Line 140 checks to

see if a two-character code has been sent; if not, the program jumps back to the key test.

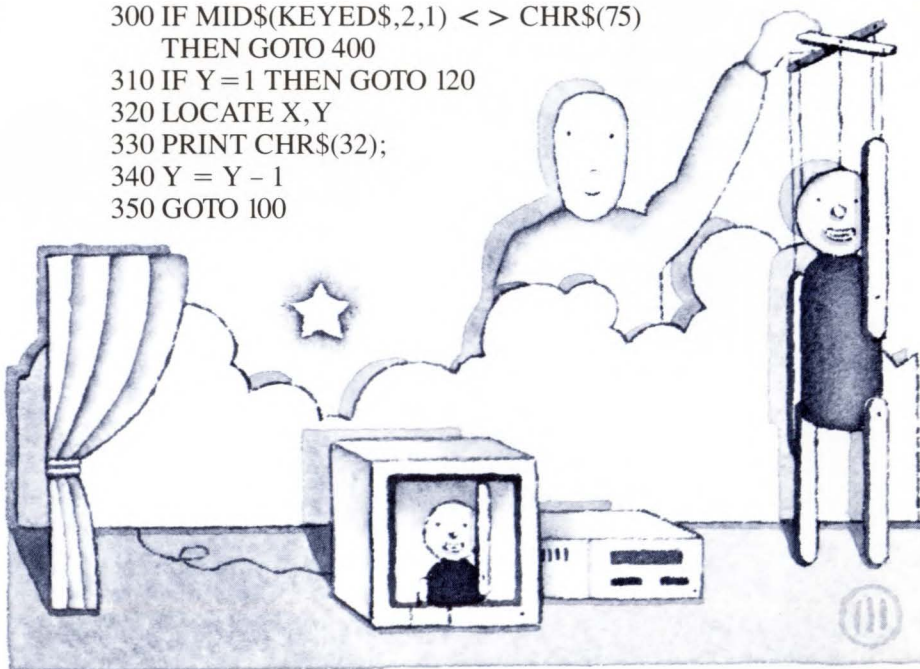
```
140 IF LEN(KEYED$) < > 2 THEN
    GOTO 120
```

The next section checks to see if the up arrow key was pressed. If not, the program jumps to the next test. If yes, the program checks to see if the character is at the top of the screen. If yes, the program considers this a no-operation and jumps back to test for another key. However, if the character is not at the top of the screen, the program first blanks out the current position and subtracts 1 from the current row; then the program jumps to the instruction that displays the character in its new position.

```
200 IF MID$(KEYED$,2,1) < > CHR$(72)
    THEN GOTO 300
210 IF X=1 THEN GOTO 120
220 LOCATE X,Y
230 PRINT CHR$(32);
240 X = X - 1
250 GOTO 100
```

Similarly, the next section tests for a left arrow key. For this key the program decrements the column rather than the row.

```
300 IF MID$(KEYED$,2,1) < > CHR$(75)
    THEN GOTO 400
310 IF Y=1 THEN GOTO 120
320 LOCATE X,Y
330 PRINT CHR$(32);
340 Y = Y - 1
350 GOTO 100
```



The next section handles the right arrow key. It increments the column unless the column was already 40, the right edge of the screen.

```
400 IF MID$(KEYED$,2,1) < > CHR$(77)
    THEN GOTO 500
410 IF Y=40 THEN GOTO 120
420 LOCATE X,Y
430 PRINT CHR$(32);
440 Y = Y + 1
450 GOTO 100
```

The next section checks for the down arrow key. If another key was pressed, control is returned to the key test routine. If the down arrow key was pressed, the row is incremented unless it was already 25.

```
500 IF MID$(KEYED$,2,1) < > CHR$(80)
    THEN GOTO 100
510 IF X=25 THEN GOTO 120
520 LOCATE X,Y
```

```
530 PRINT CHR$(32);
540 X = X + 1
550 GOTO 100
```

This BASIC program is intended to be compiled by the BASIC Compiler. After you have entered the program using an editor, you can compile it with the following command:

```
BASCOM GAME/O;
```

The /O specifies that you won't be using the BASIC Runtime Module.

Now let's assume you have given the name LOAD.ASM to the assembly language subroutine listed above, and that you have assembled it using the Macro Assembler.

After you finish your BASIC compilation, you can link the two programs with the command:

```
LINK GAME+LOAD;
```

Robot Simulation in BASIC Using the DRAW Command

*John Schnell
New York Personal Computer, Inc.*

The BASIC DRAW command and the joystick interface provide advanced interactive graphics capabilities. To demonstrate this, I have written a graphics program that uses the joystick for interactive control of a "robot arm." The positions of the upper arm, lower arm and hand are moved by pushing the joystick left or right. Each arm part can be selected to articulate individually by pressing the top joystick fire-button. You clear the screen by pressing the front fire-button. The articulating arm segment is drawn in yellow, and

each move of the arm leaves a trail of previous positions in green.

You can change individual program variables to have the robot arm do different things; for instance, you can change the length of each arm section, or change the color variable BAK = 0 so that the old arm position is erased after each move. In fact, simply by changing a few variables and program lines, this simulation can be converted into a nice program for generating graphics patterns.

Editor's note: We have placed Mr. Schnell's BASIC program ROBOTARM.BAS on the IBM PC User Group Support Electronic Bulletin Board System in the <F>iles section. For information about accessing our bulletin board system and downloading these files, see the article "IBM PC User Group Support Electronic Bulletin Board System" in the August 1985 issue of Exchange. Dial (305) 998-EBBS. If you do not have a modem, ask your group librarian to download the ROBOTARM.BAS file and make it available in your user group's library.

Keyboard Input for BATch Files

Neil Rubenking
San Francisco PC Users Group

A very simple, very tiny program (8 bytes) that you can construct using DOS DEBUG will allow your BATch files to act upon keyboard input immediately. This allows easy creation of convenient DOS-level menu systems. I read about Jacques Bensimon's GETKEY.COM file in the March 19, 1985 "User-to-User" column in *PC Magazine* and immediately applied it to my office's hard-disk menu system.

```
A> debug getkey.com
File not found (Ignore this message)
-a100
-xxxx:0100 MOV AH,00 <CR>
-xxxx:0102 INT 16 <CR>
-xxxx:0104 MOV AH,4C <CR>
-xxxx:0106 INT 21 <CR>
-xxxx:0108 <CR>
-rdx <CR>
0000
:0008 <CR>
-w <CR>
Writing 0008 bytes
-q <CR>
A>
```

Figure 1. DEBUG Script to Create GETKEY.COM
(Reprinted from *PC Magazine*, March 19, 1985, Copyright 1985, Ziff-Davis Publishing Company.)

Using IF ERRORLEVEL

The unexplored BATch command that is exploited here is "IF ERRORLEVEL ##". This condition is true if ## is less than or equal to the ERRORLEVEL. And there is a DOS function that will set the ERRORLEVEL to a given number.

Follow the DEBUG script in Figure 1 to create GETKEY.COM. The first two lines take a keystroke and put its ASCII value in register AL, and the second two put the contents of AL into the ERRORLEVEL. (But you don't even have to understand that to create the program.)

Type in the highlighted text. <CR> means press Enter.

Rules for Menus

Figure 2 shows an example of a BATch file menu using GETKEY. There are a few important points to follow when constructing such a menu:

1. There should be a label name, such as CHOOSE, just before the call to GETKEY, and another label, like BEGIN, at a point just before the menu choices are displayed.
2. The IF ERRORLEVEL statements must be in descending numerical order and should ideally be a continuous sequence. If they cannot be continuous, fill in the "holes" with IF ERRORLEVEL ## GOTO CHOOSE.
3. Put a statement just before the highest in the sequence that says "IF ERRORLEVEL (highest number + 1) GOTO CHOOSE". This traps any "irrelevant" keys that are higher in ASCII value than the real choices.
4. Put a statement just after the sequence that simply says "GOTO CHOOSE". This catches irrelevant keys that lower ASCII values.
5. Each segment of code that you transfer control to should end with "GOTO BEGIN" (except the segment that exists to DOS).

Of course, any segment of your BATch file can itself be another menu—as many levels as you like.

```
ECHO OFF
CLS
:BEGIN
ECHO _____GAMES_____
ECHO 1 SPACE PERVADERS
ECHO 2 PUNK MAN
ECHO 3 SCAR TRAK
ECHO 4 EXIT TO DOS
ECHO Press any number key to choose
ECHO _____
:CHOOSE
GETKEY
IF ERRORLEVEL 53 GOTO CHOOSE
IF ERRORLEVEL 52 GOTO EXIT
IF ERRORLEVEL 51 GOTO TRAK
IF ERRORLEVEL 50 GOTO PUNK
IF ERRORLEVEL 49 GOTO SPACE
GOTO CHOOSE
:SPACE
BASICA PERVADERS
GOTO BEGIN
:PUNK
PUNKMAN
GOTO BEGIN
:TRAK
BASIC SCARTRAK
GOTO BEGIN
:EXIT
CLS
```

Figure 2. Sample GETKEY Menu

Trapping Function Keystrokes

This is a major boon, but it wasn't quite enough for me, so I extended GETKEY.COM into GETFUN.COM. GETFUN catches all the keystrokes that GETKEY doesn't—mainly the FUNCTION keys.

If an ordinary key is pressed, it returns 255. Figure 3 shows the DEBUG script to create GETFUN.COM, and Figure 4 shows a list of the values returned for various keys. Since you can use each function key "straight" or with <Ctrl>, <Alt>, or <Shift>, a function key menu could easily have 40 different choices!

Type in the highlighted text. <CR> means press Enter.

```
A > debug getfun.com
File not found                                (Ignore this message)
-a100
-xxxx:0100 MOV AH,00 <CR>
-xxxx:0102 INT 16 <CR>
-xxxx:0104 CMP AL,00 <CR>
-xxxx:0106 JZ 010C <CR>
-xxxx:0108 MOV AL, FF <CR>
-xxxx:010A JMP 010E <CR>
-xxxx:010C MOV AL,AH <CR>
-xxxx:010E MOV AH,4C <CR>
-xxxx:0110 INT 21 <CR>
-xxxx:0112 <CR>
-rcx <CR>
0000
:0012 <CR>
-w <CR>
Writing 0012 bytes
-q <CR>
A >
```

Figure 3. DEBUG Script to Create GETFUN.COM

Function Keys		
F1.....F10	(unshifted)	59..... 68
F1.....F10	< shift >	84..... 93
F1.....F10	< Ctrl >	94.....103
F1.....F10	< Alt >	104.....113

Keypad Keys		
	(unshifted)	(Ctrl)
Home	71	119
Up	72	
PgUp	73	132
Left	75	115
Right	77	116
End	79	117
Down	80	
PgDn	81	118
Ins	82	
Del	83	

< Alt > + Regular Key		
QWERTYUIOP	16..... 25	
ASDFGHJKL	30.....38	
ZXCVBNM	44.....50	
1234567890—=	120.....131	

Figure 4. Special Keyboard Codes

Once you've created your GETFUN.COM file, you can build BATCH files with menus that prompt for and act on user input, similar to the one shown in Figure 2. With the GETFUN.COM file, the ERRORLEVELs you specify can include the extended ASCII codes. Keep in mind that the same rules apply to BATCH files that use the GETFUN.COM program as apply to BATCH files that use the GETKEY.COM program.

IBM Personal Computer AT Serial Port Pin Configuration

Steven Mahlum
IBM Corporation

The Serial/Parallel adapter for the IBM Personal Computer AT has a standard I/O parallel port, but the serial port is a new nine-pin male connector. The standard RS-232 cables for serial devices cannot be connected to the Serial/Parallel

adapter without first converting the new configuration to the RS-232 configuration.

Following is a diagram of the pin assignments for both ends of a conversion cable:

Personal Computer AT Serial Conversion Cable		
PC AT 9 pin		RS-232
1	Carrier Detect	8
2	Receive Data	3
3	Transmit Data	2
4	Data Terminal Ready	20
5	Signal Ground	7
6	Data Set Ready	6
7	Request to Send	4
8	Clear to Send	5
9	Ring Indicator	22

Puzzler

Walter Penney
Capital PC User Group

"It's an interesting problem," said Al, straightening up from the flow chart he had been working on. "This may not be the best way to go about it, but it ought to get the answer—at least to four decimal places."

"What problem is this?" asked John.

"Well, it's part of the homework in our programming course. We have to find the range of N for which $[N] + [N^2] = [N^3]$, where [N] is the greatest integer in N. Here's how I'm going to go about solving it," Al said, pointing to the flow chart.

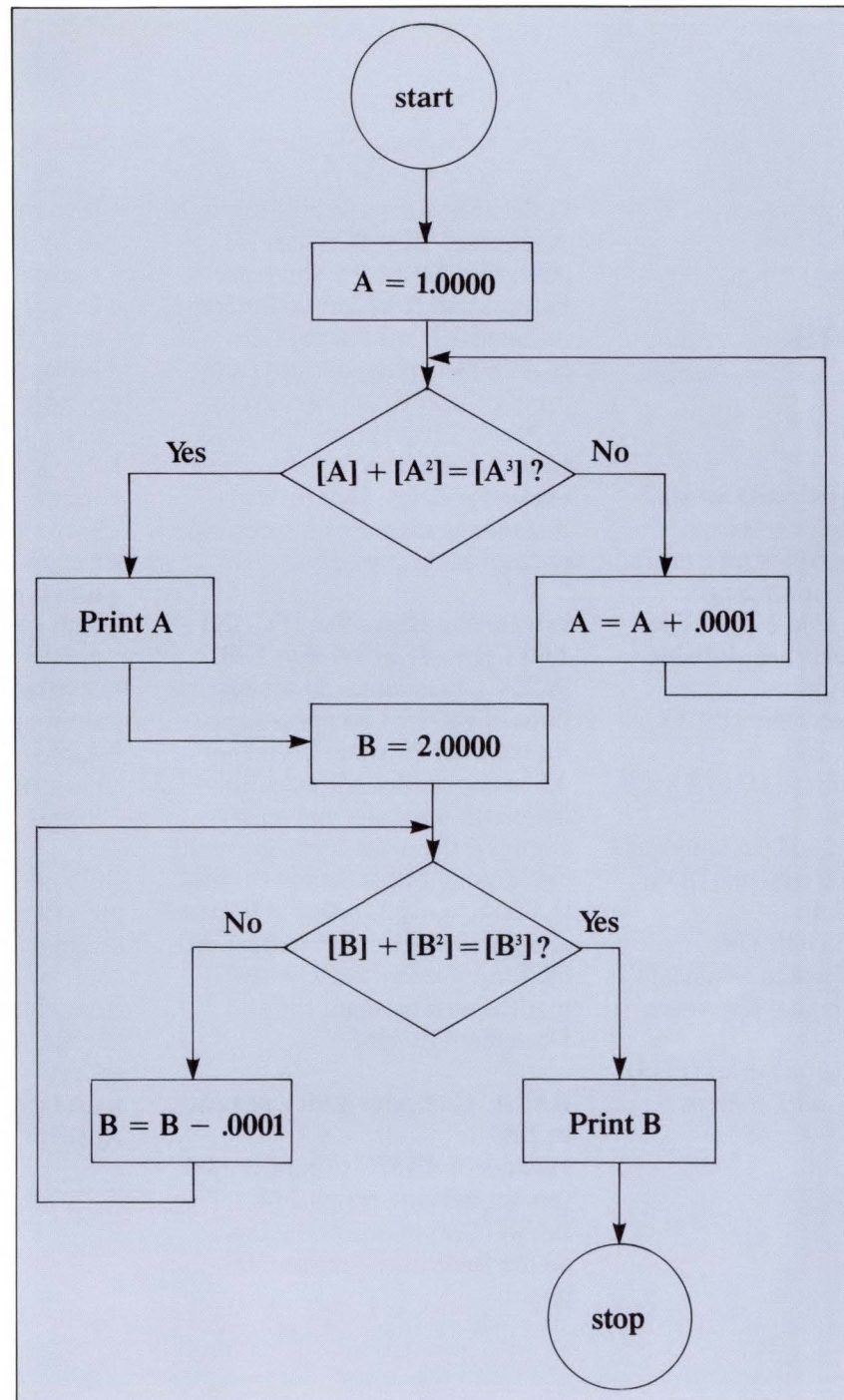
"The required range of N will then be from A to B," Al went on.

John studied it a few moments. "It looks good, but I don't think it will do what you expect," he said.

"Why not? If I work up from 1, which is too small, I ought to hit the lower limit, and if I work down from 2, which is too large, I ought to hit the upper limit. Right?"

"Perhaps, but I think you're in for a disappointment."

What is wrong?



Solution to Puzzler
The required 'range' is in two parts, from N equal to the square root of 2 to N equal to the square root of 2, and from N equal to the cube root of 3 to N equal to the cube root of 4. The values in between the square root of 2 and the cube root of 3 are not part of the solution.

IBM Product Upgrades

*John Warnock
IBM Corporation*

IBM offers upgrade kits for six Personal Computer software products. These offers let owners of existing IBM products get newer versions at less cost. The following products are available for upgrade:

- Interactive Executive (PC/IX) 1.00 to PC/IX 1.10
- BASIC Compiler 1.00 to BASIC Compiler 2.00
- DisplayWrite 2 to DisplayWrite 3
- Disk Operating System (DOS) 3.00 to DOS 3.10
- DOS 2.10 and DOS 3.00 Technical Reference Manual to DOS 3.10 Technical Reference Manual
- Personal Decision Series (PDS) Plans Edition to PDS Plans + Edition

Order forms are available through Authorized IBM PC Dealers, Authorized IBM PC Software Dealers and IBM Marketing Representatives. All six upgrades require fees (plus sales tax) and proof of license of your current software. Order envelopes must be postmarked on or before their expiration dates. Details are included in the following descriptions of each upgrade.

Interactive Executive (PC/IX) 1.00 Upgrade to Version 1.10

PC/IX 1.10 includes all the features of PC/IX 1.00 plus support for the IBM Personal Computer AT in compatibility mode. To obtain the upgrade, you must complete the order form and mail the original Maintenance Diskette [LY20-6261-0] as proof of license, along with \$40, to the address on the form. Orders must be postmarked no later than December 31, 1985.

BASIC Compiler 1.00 Upgrade to 2.00

Owners of BASIC Compiler 1.00 can upgrade to version 2.00. BASIC Compiler 2.00 includes all the features of version 1.00, plus:

- Better program control structures
- Expanded graphics capabilities
- Larger program compilation through separate data/instruction spaces
- Support for large numeric dynamic arrays
- File locking and unlocking
- Indexed Sequential Access Method (ISAM) files
- Enhanced event trapping for timer, play and key instructions
- Path specification for devices or files
- Separately compiled BASIC subroutines
- Alphanumeric labels for branching (line numbers no longer required)

To obtain the upgrade, complete the order form and mail the original beige-colored inside front cover of your BASIC Compiler manual [6172246] as proof of license, along with \$195 to the address on the form. Orders must be postmarked no later than April 30, 1986.

DisplayWrite 2 Upgrade to DisplayWrite 3

If you own either version 1.00 or 1.10 of DisplayWrite 2, you can upgrade to DisplayWrite 3. DisplayWrite 3 includes the following enhancements:

- *Helps* facility for keys, commands and menus
- Use of the *Spell Aid* during creation and revision
- Footnote typing and resolution
- Automatic outlining facility
- Keystroke programming to store and replay keystrokes
- Merging PC DOS ASCII files into text for repetitive documents
- Vertical and horizontal cursor drawing

To obtain the upgrade, complete the order form and mail the original purple-colored inside front cover of your DisplayWrite 2 version 1.00 Reference manual [6361282] or the original purple-colored inside front cover of your DisplayWrite 2 version 1.10 Reference manual [6361822] as proof of license, along with \$50, to the address on the form. Orders must be postmarked no later than November 11, 1985.

Disk Operating System (DOS) 3.00 Upgrade to 3.10

Owners of DOS 3.00 can upgrade to version 3.10 in order to add support for the IBM Personal Computer Network. To obtain the upgrade, complete the order form and mail the blue-colored inside front cover of the Disk Operating System (DOS) 3.00 Reference manual [6322666] as proof of license, along with \$30, to the address on the form. Orders must be postmarked no later than December 31, 1985.

DOS 2.10 or 3.00 Technical Reference Manual Upgrade to 3.10

The DOS 3.10 Technical Reference manual includes all the information covered in the DOS 2.10 and DOS 3.00 Technical reference manuals plus information about DOS 3.10. The new manual also includes an operational Update Information Service to keep your manual up-to-date automatically.

Send in the upgrade order form plus either the beige-colored inside front cover of the DOS 2.10 Technical Reference manual [1502346] or the blue-colored inside front cover of the DOS 3.00 Technical Reference manual [6322677], along with \$65, to the address on the form. Orders must be postmarked no later than December 31, 1985.

Personal Decision Series (PDS) Plans Edition Upgrade to Plans +

PDS Plans + Edition is an enhanced version of PDS Plans Edition that adds:

- Advanced functions for financial, statistical and engineering analysis
- Enhanced modeling and programming capabilities
- Consolidation or deconsolidation of spreadsheets
- New arithmetic, logic and control functions

To get Plans + Edition, complete the order form, provide the Plans Edition serial number, and send the purple-colored inside front cover of the Plans Edition manual [SH20-9549] and the Plans 1 diskette [SV30-0536], along with \$150, to the address on the form. Orders must be postmarked no later than April 30, 1986.

Editor's note: Upgrade announcements will appear periodically in Exchange to keep readers informed of the latest offerings.

User-Written Articles in *Exchange*

Reprinting Articles

Several user groups have asked us for permission to reprint articles from *Exchange* in their own group newsletters.

To satisfy these requests, we are placing selected user-written articles onto our PC User Group Electronic Bulletin Board System (EBBS) on a timely basis after they appear in *Exchange*. Articles that we place on our EBBS will be those that were written by members of user groups for their own group newsletters and then published in *Exchange*. You may freely copy and reproduce any articles we place on our EBBS.

Articles in *Exchange* that were written by IBM will not be placed onto the EBBS, and will appear only in printed form in *Exchange*, which is a copyrighted publication. Please respect the copyright and do not reproduce IBM-written articles in your own group newsletters.

A Call for Fresh Articles

We are placing user-written articles onto our EBBS in order to satisfy user group newsletter editors who need material to publish. Whereas this helps satisfy a need, the result may be that many user group newsletters will carry the same material, and user group members will not be encouraged to write fresh articles for their group newsletters.

In recent months we have seen a decrease in the number of fresh articles in user group newsletters we have been receiving. The vitality and continuing success of newsletters—both your own group's newsletter and *Exchange*—depends on a continual flow of new articles. We in IBM will continue to write our own new material for *Exchange*, but to attain a proper balance, we need new material from user group newsletters as well.

This is a call for fresh articles about IBM Personal Computer hardware and software. I encourage you to develop and publish good and valuable PC articles and information for your own group newsletter. Then, please be sure that your group is sending us its newsletter, so that we can see your article.

We would like to receive newsletters from user groups that are not yet sending newsletters to us. We make this request for two reasons: (1) We are confident that user group newsletters we do not yet receive will contain fresh articles equally as good as those we have received thus far. (2) We are considering distributing *Exchange* to only those user groups that send us their newsletters.

Please send your newsletters to us at:

IBM PC User Group Support
IBM Corporation (2900)
P.O. Box 3022
Boca Raton, FL 33431-0922

Thank you.

Michael Engelberg
Editor

Copyrights, Trademarks and Service Marks

ColorPaint by Marek and Rafal Krepec Incorporated.

ColorPlus is a trademark of Plantronics Corporation.

CompuServe is a trademark of CompuServe, Incorporated.

CP/M is a registered trademark of Digital Research, Incorporated.

CP/M-86 is a trademark of Digital Research, Incorporated.

Data Encoder and its associated documentation are under the U.S. Department of State Munitions list, Category XIII(b) and, as such, must be licensed by the U.S. Department of State prior to export from the United States.

DIF is a trademark of Software Arts, Incorporated.

Dow Jones News/Retrieval Service is a registered trademark and Dow Jones is a trademark of Dow Jones & Company, Incorporated.

EasyWriter is a trademark of Information Unlimited Software, Incorporated.

Electric Poet is a registered trademark of Control Color Corporation.

Fact Track is a trademark of Science Research Associates, Incorporated.

HomeWord is a trademark of Sierra On-Line, Incorporated.

IBM is a registered trademark of International Business Machines Corp.

INTERACTIVE and IS/5 are trademarks of Interactive Systems Corporation.

Jumpman is a trademark of EPYX, Incorporated.

King's Quest is a trademark of Sierra On-Line, Incorporated.

Logo is a trademark of Logo Computer Systems Incorporated.

Lotus and 1-2-3 are trademarks of Lotus Development Corporation.

Managing Your Money is a trademark of MECA (TM).

MECA is a trademark of Micro Education Corporation of America, Incorporated.

Microsoft and the Microsoft logo are registered trademarks of Microsoft Corporation.

Multiplan is a U.S. trademark of Microsoft Corporation.

NEC is a trademark of Nippon Electric Co., Ltd.

PCjr is a trademark of International Business Machines Corp.

PC Mouse is a trademark of Metagraphics/Mouse Systems.

Peachtext is a trademark of Peachtree Software Incorporated, an MSA company.

Personal Computer AT is a trademark of International Business Machines Corp.

Personal Computer XT is a trademark of International Business Machines Corp.

pfs: is a registered trademark of Software Publishing Corporation.

PlannerCalc is a trademark of Comshare.

REALCOLOR is a trademark of Micro Developed Systems, Inc.

SHAMUS is a trademark of SynSoft(TM).

SMARTMODEM is a trademark of Hayes MicroComputer Products, Inc.

Synonym information in PCWriter and Word Proof is based on the American Heritage Dictionary Data Base, Roget's II, The New Thesaurus, owned by Houghton Mifflin Company and used with permission. Copyright 1982 by Houghton Mifflin Company.

The Learning Company reserves all rights in the Rocky, Bumble, Juggles and Gertrude characters and their names as trademarks under copyright law. Rocky's Boots, Bumble Games, Bumble Plot, Juggles' Butterfly, Gertrude's Puzzles, Gertrude's Secrets and The Learning Company are trademarks of The Learning Company.

THE SOURCE is a service mark of Source Telecomputing Corporation, a subsidiary of The Reader's Digest Association, Incorporated.

Time Manager is a trademark of The Image Producers, Incorporated.

TopView is a trademark of International Business Machines Corp.

UCSD, UCSD p-System and UCSD Pascal are trademarks of the Regents of the University of California.

UNIX is a trademark of AT&T Bell Laboratories.

VisiCalc is a trademark of VisiCorp.

Visi On is a trademark of VisiCorp.

WD212-X is a trademark of Wolfdata, Inc.

Word is a U.S. trademark of Microsoft Corporation.

WordStar is a trademark of MicroPro International Corporation.

XENIX is a trademark of Microsoft Corporation.

Z-80 is a registered trademark of Zilog.

“ In BASIC Compiler 2.00, you can use the BASIC Interpreter to run and debug programs and later compile those programs to increase their execution speed. (page 1)

“ IBM's new PC Storyboard is almost as much fun to use as gameware, but the resulting output is seriously useful in business. (page 7)

“ IBM PC Storyboard could renew home interest in PCs. It could turn into a whole new fun-and-profit hobby all by itself. (page 8)

“ You can create a file to effect key reassignments directly from the keyboard using the COPY command and the PROMPT command. (page 11)

“ You can tell what subdirectory you are in and what time it is by customizing the DOS PROMPT command. (page 22)

“ In the graphics screen modes, it is relatively easy to define any or all of the uppermost 128 ASCII characters so that they become your own customized graphics characters. (page 27)

“ Since you can use each function key “straight” or with <Ctrl>, <Alt>, or <Shift>, a function key menu could easily have 40 different choices! (page 32)