# On the Way to Standard BASIC

## *A survey of what's in the proposed ANSI standard and why it's there.*

**Thomas E. Kurtz**
**Dartmouth College**
**Hanover, NH 03755**

The American National Standards Institute (ANSI) committee X3J2, charged with developing a standard for the BASIC programming language, held its first meeting in January 1974. We're now well into the eighties and still we have no published standard. Why so long? The standardization process is at best slow and cumbersome, but need it be this slow? After all, standards for FORTRAN, COBOL, and PL/I have been around for a while. Half of the short answer is that the X3J2 committee has produced in eight years two standards: the "draft," which is the subject of this article, and Minimal BASIC, which appeared in 1978 (see reference 1). (Minimal BASIC has not caught on because the rapid development of chip technology has made its modest capabilities obsolete.) The other half of the short answer is that BASIC was changing drastically while the committee was trying to standardize it. In otherwords, BASIC has been a moving target.

BASIC got its start as a simple language designed to make life easier for the nonexpert programmer. But what started out as a simple language with no more than a dozen different statements dealing only with numbers has grown into a diverse language with many statements capable of handling numbers, strings, arrays, files, and plotting. Single-letter and letter-digit variable names have grown to multicharacter variable names. Simple GOTOs and IF ... THENs have evolved into the famous constructs of structured programming. Each vendor has developed its own formats and rules for these extensions so that present versions of BASIC differ widely as to form and content.

At long last, the diverging paths are being brought together in the new proposed standard for BASIC. The standard includes structured constructs, a 'MAT" (matrix) package, formatted output, subprograms that can be made independent, files, exception handling, and optional sections on graphics, sophisticated file structures, real time, fixed decimal arithmetic, and editing. (Details appear later in this article.) The example in listing 1 illustrates a few of the highlights of standard BASIC: multicharacter identifiers, subprograms, and several of the structured constructs.

The standard is currently (early 1982) under technical committee mail ballot, which will assert (if it passes) that the technical development of the standard has been completed. The next major milestone is a public-review period conducted by ANSI's X3 committee, which supervises the making of all computer-related standards in the United States. Further steps, which usually take several years to complete, will follow before the standard becomes ANSI official, but these are formalities that have little likelihood of changing the content of the standard.

The standard will mainly benefit the educational world. Programs published in magazines such as BYTE may eventually be in standard BASIC rather than in some variation. Textbooks containing programs won't have to be written specifically for a particular brand or model of computer. Finally, programs written in standard BASIC will be easier to transport and distribute.

### Difficulties with Standardization

Developing a standard for BASIC has been difficult because the language serves such a diverse clientele.

Educational users tend to work on mini- and microcomputers. They desire a language that is easy to leam and is not cluttered with declarations or excessive structure. They would be satisfied with fairly simple file systems.

**About the Author**

*Thomas E. Kurtz is co-author of the original, "Dartmouth" BASIC and is chairman of the ANSI committee that has developed the proposed draft standard for BASIC.*

**Listing 1:** *This program incorporates many of the new features of the proposed draft standard for BASIC.*

```
100 Program CRAPS
110 !
120 ! A simple program in standard BASIC
130 !
140 ! Plays N games of craps
150 !
160 ! Read n
170 Data 10
180 !
190 For i = 1 to n
200         Call DICE (Total)
210         Print "You rolled a "; Total
220         Select Total
230         Case 7, 11
240               Print "You win."
250         Case 2, 3, 12
260               Print "You lose."
270         Case else
280               Print "which is your point."
290               Do
300                     Call DICE (Newtotal)
310                     Print Newtotal,
320               Loop until Newtotal = 7 or Newtotal = Total
330               If Newtotal = 7
340               Then
350                       Print "You lose."
360               Else
370                       Print "You win."
380               End if
390 End select
400 Next i
410 !
420 End
430 !
440 !
450 Sub DICE(Sum)
460 !
470 ! Roll two dice and add them up
480 !
490 Let dl = Int(6*Rnd + 1)
500 Let d2 = Int(6*Rnd + 1)
510 Let Sum = dl + d2
520 !
530 Sub end
```

Another group of users includes those with large machines or with access to large machines. These users want a rich, compiler-based language. They want to construct subroutine libraries of independently compiled subprograms. This group also wishes to write interactive programs that process strings of characters, something that FORTRAN and COBOL don't do easily. (PL/I allows string processing, but it's not accessible in many interactive environments.) Pascal does not offer what these users want either; it is too pristine.

A third group of users wishes to do business and financial calculations using BASIC. Such use is extensive partly because many of the early financial applications were written in BASIC. In Europe, BASIC is the primary business data-processing language for small computers. This group wants formatted output, accurate dollars-and-cents calculations, and access to record-structured files. Few suitable alternatives exist for these users on small machines.

### What Most BASICs Are Like

Present-day BASICs, including the current version of the draft standard, reflect most of the goals of the original version of BASIC. For example, most BASICs avoid declaration of variables, with the notable exception that many, including the draft

standard, require declaration of *arrays* (lists and tables). In most BASICs, variables are typed implicitly, according to some special symbol. Thus, string variables have the dollar sign ($) in their name. This convention limits the number of different types of variables, because there aren't many special characters left. Some argue that this is good, not bad.

It is still true that a small job requires only a small program. Some BASICs even allow omitting the END statement. A language that lacks declarations and excessive structure lends itself more readily to interpreters. For these, a simple computation requires but a single statement. If you want to add 2 and 2, the single direct statement:

```
PRINT 2 + 2
```

will work. Try this in Pascal or FORTRAN!

Because a user can get by with a minimum of syntax rules and structure, BASIC is easy for novices to learn. Perhaps even more important, it is easy for occasional users to remember. I know teachers who use the computer only twice a year but who can remember what to do without having to check the manual.

**The Proposed Standard**

The standard will of course embrace most of simple BASIC or Minimal BASIC. (ANSI Minimal BASIC is similar to the earliest versions of BASIC. It includes the REM, LET, INPUT, PRINT, READ, RESTORE, DATA, DIM, FOR, NEXT, IF. . . THEN, line number, GOSUB, GOTO, RETURN, ON. . .GOTO, RANDOMIZE, single-line DEF, STOP, and END statements. It lacks string lists, files, plotting, etc. ANSI Minimal BASIC is quite minimal!) BASIC extends Minimal BASIC in a number of ways, for example, by allowing multicharacter variable names. It also includes features completely missing from Minimal BASIC, such as graphical output and real time. Incidentally, the committee elected to use the name BASIC for this standard. It had used the terms "Extended BASIC" and "Enhanced BASIC," but it dropped the modifiers, thus allowing their use by vendors later.

The standard is written so as to define *standard-conforming programs*. Any program that is written according to the rules of the standard is standard-conforming. A standard-conforming *implementation* (interpreter or compiler) is one that will correctly process a standard-conforming program. A standard-conforming implementation may offer extensions, provided that all standard-conforming programs will continue to be correctly processed. This point is important in order to understand some of the choices made by the committee.

Actually, the standard will consist of a core module plus five optional modules: enhanced files (direct access and keyed); graphics; real time; fixed decimal (for business users); and editing.

I'll now give a section-by-section summary of the features of the proposed standard.

**Data Types**

BASIC includes variables and constants of type numeric and string. Numeric is, of course, single precision. The standard will not specify other types, such as integer or double precision, both of which have been requested by part of the user community. It will provide fixed decimal but only as an option. The reason for not including other types is that BASIC serves many masters—large machines, small machines, microcomputers, interpreters, compilers, education, business—making the choice of data types difficult. As it is, vendors can enhance their own versions of the standard BASIC with whatever additional data types are needed by their users. Of course, programs written to take advantage of such data types will not be standard-conforming and might not be transportable.

**Program Comments**

In addition to the REM statement

for comments, BASIC will allow on-line comments using the exclamation point (!).

**Identifiers**

It did not take the committee long to vote for multicharacter variable names. Up to 31 characters (letters, digits, and underlines, starting with a letter) are permitted for variable and function names, with the trailing dollar sign for string-variables counting. Despite the obvious advantages over old-fashioned BASIC variable names, multicharacter names exact their price. For instance, spaces are required around keywords (such as FOR N = 1 to M instead of FORN = lTOM), and certain words cannot be used as variable names (mainly, the names of the functions with no arguments, such as RND). The first restriction is probably a good idea anyhow. The second poses a challenge to implementers to come up with reasonable error messages when unsuspecting novice users try to assign one of these words as a variable name.

Incidentally, uppercase and lowercase may be used interchangeably for keywords, identifiers, function names, etc. Of course, the cases remain distinguishable in quoted strings, input replies, etc.

**Numeric Operations**

The big news is that arithmetic will be floating decimal. Thus, 2.29 + 4.71 = 7.00, not 6.999999. Also:

.1 + .1 + .1 + .1
+ .1 + .1 + .1 + .1
+ .1 + .1 = 1 *exactly*

Vendors may offer native arithmetic (presumably floating binary or floating hexadecimal) as an option if efficiency is an issue. But users will finally be able to carry out dollars-and-cents calculations with confidence.

Minimal BASIC provides these numeric functions: ABS, ATN, COS,

*EXP*, INT (the floor), LOG (natural), RND, SGN, SIN, SQR, and TAN. New numeric functions include ACOS, ANGLE (easier to use than ATAN for determining the angle given the base and height), ASIN, CEIL (ceiling, the opposite of INT), COSH, COT, CSC (cosecant), DATE, DEG (radians to degrees), EPS (the smallest representable positive number), FP (fractional part—the same as X-INT(X) for positive X), INF (the largest positive number), IP (integer part—the same as INT for positive values), LOGlO, LOG2, MAX, MIN, MOD, PI, RAD (degrees to radians), REM (remainder—the same as MOD for positive numbers), ROUND, SEC, SINH, TANH, TIME, TRUNCATE (reduce the number of significant digits, but don't round).

Taking a cue from the hand-held calculators, the user at his option can express angles in degrees instead of radians. Secondary school trigonometry and general math students should benefit because normally students don't learn radian measure until they take calculus.

**String Operations**

The two important operations on strings are concatenation (joining two strings) and substring extraction. The former is accomplished by using the ampersand (&); the latter is achieved by following the string variable with a range enclosed in parentheses. Thus, LINE$(4:7) gives the fourth through seventh characters of the string LINE$. This substring notation can appear on the left side of a LET statement, in which case the fourth through seventh characters are replaced by whatever appears on the right side of the LET statement. The old substring functions (SEG$, MID$, LEFT$, etc.) that we have come to know and love are gone. Good riddance!

Functions whose arguments or values are strings include CHR$, DATE$, LEN, LCASE$, ORD, POS, STR$, TIME$, UCASE$, and VAL. CHR$ and ORD are opposite, and associate a character to its numerical value. LCASE$ and UCASE$ are lowercase- and uppercase-conversion functions. DATE$ and TIME$ give the date and time as strings. STR$ and VAL are the number-string conversion functions that have been around for a while in BASIC. LEN gives the length of a string. POS searches a string for another string.

**LET Statement**

This brings us to a point that may disturb many. The LET in the LET statement is mandatory! One might ask why, as the option to omit it is such a common convention and a convenience to many users. The principal reason was to reduce the number of reserved words while retaining simple parsing. As it is, only the no-argument function names like RND and the words NOT, PRINT, REM, and ELSE are reserved (not allowed to be used as identifiers). This means that a user cannot write LET RND = 3. One should realize, however, that the standard actually prescribes only that *standard-conforming programs* may not omit the LET. Compilers and interpreters may, however, allow users to omit the LET, but they must accept the LET when it is present. Such implementations will have to be smart enough to recognize that:

```
INPUT = 3
```

is a LET statement and not an input statement.

**Arrays**

Arrays must be dimensioned in the program before use. This rule conflicts with Minimal BASIC, which allows default dimensions for lists and tables (vectors and matrices) that do not appear in DIM statements. You might wonder why we are doing away with the convenience of not having to dimension small arrays. The reason is that general identifiers are now allowed for both arrays and functions. For example, in "LET X = A(3)" the meaning of A(3) is

ambiguous because it could be either element 3 of the array A, or the function A evaluated at 3.

There are three ways out of the dilemma. First, a two-pass compiler (or intepreter that does a pre-scan) could assume that A(3) was an array if it didn't find a function definition for A later in the program. Second, one could require that all functions be declared early in the program. Third, one could require that all arrays be declared early in the program. The committee adopted the third option because most arrays have to be dimensioned anyway and it's customary to place the dimension statements early in the program.

A MAT package includes matrix (or vector) input and output, scalar multiplication, matrix add, subtract, and multiply, and the matrix functions of linear algebra INV, TRN, DOT, and DET. Even those who have no interest in linear algebra will find the MAT input and output statements handy. For instance, suppose a small firm has several departments, and that the sales results for all of them are kept in several lists (one-dimensional arrays). Then:

```
MAT net_sales =
        gross_sales - expenses
```

will calculate the net sales for all departments at once. (The above statement, and others like it later in the article, is intended to occupy a single line.)

Only one- and two-dimensional arrays are included in the standard, though designers of interpreters and compilers may choose to allow more.

**Logical Expressions**

Minimal BASIC allows only simple relational expressions (such as X < = Y) in IF statements. BASIC allows these to be combined using AND, OR, and NOT to form logical expressions. Parentheses are allowed, in case you forget whether AND takes precedence over OR or vice versa. Whereas Minimal BASIC allows only = and < > with strings (as with IF A$ = "YES"), BASIC allows the full range of relational operators with strings. What actually happens when "IF A$ < B$" is used depends on the collating sequence. For instance, the ASCII collating sequence specifies that "B" comes *before "a"*.

**Branching and Decision Making**

The programmer can continue to use GOTO and IF. . .THEN from Minimal BASIC. Or instead he can choose to use structured constructs now typical of almost all programming languages. Take, for instance, the IF. . .THEN. . .ELSE construct. In BASIC, this takes the form:

```
IF <logical expression> THEN

ELSE

END IF
```

Two important features of this construct are, first, the keywords that define the construct must appear at the beginning of separate lines. Thus, the ELSE and END IF cannot be obscurely buried near the end of a line. Second, the construct ends with a keyword sequence that is unique to that construct.

You can use the simple one-line IF. . .THEN. . .ELSE, which might look like this:

```
IF x < y THEN LET a = 3
   ELSE LET a = 4
```

With both forms of IF. . .THEN . . .ELSE, the programmer may omit the ELSE part.

**Looping Structures**

The FOR NEXT loop of Minimal BASIC is retained, and a new structure, the DO LOOP, is added. The loop-ending condition (or conditions) may be attached to the DO statement, the LOOP statement, or both. The loop-ending condition may be expressed either as a WHILE or as an UNTIL. The following is typical:

```
DO UNTIL i > n OR a$ = list$(i)

LOOP
```

In addition, a DO LOOP may be *exited* with an EXIT LOOP statement. Whenever such a statement appears in the body of a loop, the next statement executed will be the one following the first LOOP statement encountered. The following example is typical:

```
DO
   PRINT "Input an integer ";
   PRINT "between 1 and 7";
   INPUT x
IF 0 < x AND x<=7 AND
      x = INT(x) THEN EXIT DO
   PRINT "Bad number; reenter"
LOOP
```

The EXIT DO gets you out of a DO LOOP. Similarly, an EXIT FOR gets one out of a FOR NEXT loop. The previous example of the DO LOOP for searching a string list could also be written:

```
FOR i = 1 TO n
   IF a$ = list$(i) THEN EXIT FOR
NEXT i
```

Exit statements are also provided for multiline defined functions and subprograms.

**Multiway Selection**

A SELECT construct allows choosing one of many alternatives. The following example illustrates some of its features:

```
SELECT DICE
CASE 7, 11
   PRINT 'Win"
CASE 2, 3, 12
   PRINT "Lose"
CASE ELSE
   PRINT "Roll again"
END SELECT
```

**Functions, Subprograms, and Chaining**

Minimal BASIC gives us two simple methods for program modularization—single-line defined functions and subroutines (of the GOSUB RETURN type). BASIC adds three methods: multiple-line defined func-

tions, subprograms, and the ability to chain. Multiple-line defined functions begin with a DEF statement and end with an END DEF statement. In between, there can be any code, but there should be at least one LET statement having the name of the function on the left side.

Subprograms are extemal to the main program and to each other. Their intemal variables are thus local to them, in contrast with defined functions, which can access all variables in the program unit in which they are defined. Parameters of subprograms can be numeric, string, array (of either type), or a channel-setter (which refers to a file, which I'll discuss later). Because input to and output from subprograms is through the calling sequence only, subprograms can be separately compiled (on those systems that provide compiling) and collected into libraries.

Multiple-line defined functions may also be made external to the program, so they can be collected into libraries. In this use, they start with the keyword FUNCTION instead of DEF and end with END FUNCTION.

A CHAIN statement allows a program to stop and start running some other program, which could be in a different language. Information may be passed to the chained-to program through an argument list that works the same as with defined functions. That is, arguments may be numeric or string expressions or arrays, and they are called "by value." The corresponding parameters in the chained-to program follow the keyword PROGRAM.

**Input and Output**

The READ and DATA statements work as they do in Minimal BASIC. String data can be quoted, in which

case the string includes all characters between the quote marks including possible leading and trailing spaces. If the string data contain no leading or trailing spaces, commas, or quote marks, they may be unquoted.

The INPUT statement works as in Minimal BASIC. Inputting an entire line without regard to commas and leading and trailing spaces is done with the LINE INPUT statement.

Input-prompt strings other than the "?" can be provided. In addition, timeout control can be added. The following example is typical:

```
INPUT PROMPT "Answer = ",
      TIMEOUT 5, ELAPSED t:
      answer
```

As in Minimal BASIC, PRINT statements may use the comma to move to the next print zone, the semicolon to stay where you are, and the TAB function to move to a specified columnar position. BASIC provides, in addition, a PRINT USING statement for more elaborate output formatting. The image, which is like a picture of the eventual printed line, can be contained in a string or in an IMAGE statement referred to by its line number. The alternate forms of the PRINT USING statement are:

```
PRINT USING format$: . . .
   or
PRINT USING 100: . . .
```

Array input and output are also included. Variable amounts of input can be received by the statement:

```
MAT INPUT A(7)
```

**Files**

BASIC provides for four types of file organization: sequential, stream, relative, and keyed. Sequential files consist of records that must be accessed sequentially. Stream files consist simply of a stream of values and must also be accessed sequentially. Relative files are sometimes called random-access files; they probably will exist on disks. Keyed files are accessed not by record number but by some key.

Three types of records are described: display, internal, and native. Display records are produced by PRINT statements—strings of characters ending in a carriage return-linefeed. IntemaI records contain values of numbers or strings, but in some internal format. The key point is that what gets read back in is exactly identical to what was written out. (This is not necessarily true with display-format files, as numbers must be converted to strings of characters on output and from strings of characters back to numbers on subsequent input.)

Of the 12 combinations of file organization and record type, only 3 are required in the core standard: sequential-display, sequential-internal, and stream-internal. Five other combinations are defined as possible enhancements. The remaining 4 are not defined by the standard, which leaves open the possibility that some implementations may use them.

The OPEN statement associates a channel-setter of the form "#13" to a file whose name is given. Ways are provided to find out if a file exists, and if it does, what its attributes are. The CLOSE statement closes a file. The ERASE statement erases the contents of a file and leaves it of zero length. Two examples:

```
OPEN #infile: NAME "Myfile",
        ACCESS INPUT,
        ORGANIZATION
        SEQUENTIAL

OPEN #3: NAME "filename"
```

In the second example, it is assumed that the organization is sequential; the record-type, display; and the access, "outin" (both input and output).

PRINT and INPUT are used to pass information to and from sequential-display files, just about the way they work for the terminal. READ and WRITE are used to communicate with all three file types. Display files can thus be accessed by both PRINT and INPUT, and READ and WRITE. Native-format files are accessed through "templates" and are provided for possible access to COBOL files.

**Exception Handling**

The construct for intercepting exceptions (situations during execution that usually cause the program to terminate) is:

```
        WHEN EXCEPTION IN

        USE

        END WHEN
```

The following simple example can be used to protect against an invalid VAL argument;

```
LET flag = 0
WHEN EXCEPTION IN
   LET x = VAL(a$)
USE
   PRINT "Bad number; reenter"
   LET flag = 1
END WHEN
```

The USE part is invoked when any exception whatsoever occurs during the WHEN part. In the previous example, it is almost true that only one kind of exception is possible. The printed error message will thus be correct most of the time. The programmer may double-check by using the EXTYPE function, which returns the coded number of the exception. For the example above, the EXTYPE value is 4001. Some 144 exceptions are defined and coded in the standard.

## A CAUSE statement can force any particular exception.

More elaborate exception handlers may be constructed. For such purposes there are the RETRY, CONTINUE, and EXIT HANDLER statements and the EXLINE function. RETRY sends control to the start of the line in which the exception occurred, CONTINUE sends control to the line following that in which the exception occurred, while EXLINE has as its value the line number of the line in which the exception occurred. There is also a CAUSE statement that can force any particular exception to occur; a programmer can use the cause statement to check his exception-handling code.

**Graphics**

The language includes statements to carry out simple plotting. The basic plotting statement is PLOT. It can be used to plot dots or straight lines. As examples:

```
PLOT X,Y
PLOT X1,Y1; X2,Y2
PLOT
```

If the beam is off, the first plots a dot at (X,Y), and the second draws a line from (X1,Y1) to (X2,Y2). If the beam is on, the first and second also draw a line from the previous point to (X,Y) or (X1,Y1), respectively. The third turns the beam off (lifts the pen) if it's on and does nothing if the beam is off.

**Listing 2:** *A graphics program written in the proposed draft form of standard BASIC.*

```
100 ! Town
110 !
120 ! Draws a picture of a town
130 !
140 1 Naming the type of plotter is implementation-defined
150 !
160 Window 0, 4, 0, 3
170 !
180 Plot Town
190 !
200 End
210 !
220 Picture Town
230 !
240 For i = 1 to 2
250         For J = 1 to 3
260             Plot House with scale(.5) * shift(i,j)
270         Next j
280 Next i
290 !
300 End picture
310 !
320 Picture House
330 !
340 Plot 0,0; 0,1; 1,1; .5,1.5; 0,1; 0,0
350 !
360 End picture
```

The points to be plotted are given in user coordinates, which are specified with a WINDOW statement. The programmer may specify the physical size of the screen to be used, arrange to CLIP the picture, SET the color and line style, and use the ASK statement to find out about the current status of these quantities. The GRAPHIC INPUT and GRAPHIC PRINT statements provide text input and output. Polygon fill can be accomplished by:

MAT FILL POLYGON

Complicated pictures may be built from simple ones with PICs, which are like subprograms. They are invoked with the PLOT statement (rather than with the CALL statement). As they are plotted, transformations of various types may be made. These include SHIFT, SCALE, ROTATE, and SHEAR, and combinations thereof.

Listing 2 is a complete program for drawing a picture of a town.

**Real Time**

Most BASIC users would be surprised to learn that BASIC is one of the important languages for real-time applications, such as industrial-process control. The reason is that it is simple and can be provided easily on the small machines used in such applications. Standard BASIC will include optional features, such as parallel sections and definitions of device interfaces, to permit this use. This work grew out of earlier work sponsored by the IEEE in developing a standard for CAMAC (Computer Automated Measurement and Control) BASIC.

A real-time program consists of parallel sections, each of which is an independent program unit with respect to line numbers and identifiers. Each parallel section can receive input from and send output to any device specified and can exchange messages with other parallel sections. A section can become dor-

mant upon reaching a WAIT statement and be awakened when some specified event or condition has occurred. Within a parallel section, all the usual BASIC statements may be used, including subprograms. Of course, there must be a supervisor program behind the scenes that attends to all message passing and scheduling.

As with BASIC in general, an example program that illustrates all of the features of real-time BASIC would be prohibitively long. Just a hint can be gotten from this simple example:

```
320 PARACT RIG1
330       WAIT TIME 17*60*60
340       PRINT "Time to go home."
350 END PARACT
```

This parallel section will "hang" until 61,200 seconds have passed since midnight; it will then "wake up," print the message, and then loop back to the WAIT statement until 5 p.m. the next day.

**Fixed Decimal**

An optional fixed-decimal module allows programmers to specify that all numeric variables and expressions be fixed decimal. Of use to data-processing programs, it can be invoked by an option statement. For instance;

```
OPTION ARITHMETIC FIXED*8.2
```

specifies that fixed-decimal arithmetic is to be used and that all variables (except those declared otherwise) must permit eight digits before the decimal point and two digits after.

Individual variables may be declared to have precisions other than those prescribed in the OPTION ARITHMETIC statement by using the DECLARE statement. For instance:

```
DECLARE NUMERIC
   national__ debt*15.2
```

would allow values up to a penny less than 1 quadrillion dollars.

**Editing**

Although legally not part of the standard, an optional module will suggest forms to be used for the editing operations often associated with BASIC programs. These include LIST, EXTRACT, DELETE, and RENUMBER.

**Summary**

For several years now, BASIC has been the de facto standard-programming language for small computers (and in Europe, for business computers as well). Finally, the de facto standard is about to become *standardized.* Far from holding back innovation, the proposed draft standard will be a major force in keeping software up to pace with hardware advances in the eighties.

## References


1. *American National Standard for the Programming Language Minimal BASIC,* X3.60-1978. ANSI, NewYork, 1978.
2. Kurtz, Thomas E., “Basic," from *History of Programming Languages.* Academic Press, 1981, pp. 515-549.

## Timetable for Approval

*This article is the first public presentation of the main features of the standard now in preparation. The* **X3J2** *committee will shortly send its proposed standard to the parent committee X3. X3 will then establish a public-comment period during which copies of the proposed standard will be available. The public is then invited to examine thestandard, pointoutflaws, or propose modifications. Individual computer users and usergroups should be on the lookout for the public-comment period and respond with suggestions or comments.*

*We also hope that the trade and academic press will examine the standard when it becomes available and draw comparisons between it and other popular versions of BASIC. The X3J2 committee doesn't really expect all vendors to implement all that is in the standard. But we hope that what vendors do implement will be compatible with the standard.*

*The schedule of events in the near future for the standard is:*

**Late July 1982:** *Confirm the technical review, possibly make last-minute changes.*
**Fall 1982:** *Transmit the standard to X3 for further processing. At this point, the standard will be virtually stable, and vendors and users can begin to count on its features. Subsequent processing of the standard is largely formal, although it is possible to change the standard when there is a significant public aversion to some feature in the standard.*
**Late 1982 or early 1983:** *Public comment period and letter ballot within X3.*
**1983:** *Transmittal to ANSI for still further processing.*
**1983:** *Final approval by ANSI.*

*As with any best-laid plans, unforeseen problems can only cause delays. The above schedule is therefore optimistic. On the other hand, the technical content of the standard is not likely to change after the fall of 1982. Implementers should be able to plan new compilers and interpreters with confidence at that time.*

## How Standards Are Written

*One characteristic of standards* work is *that major points are usually settled easily, while seemingly minor points may take years to resolve. A classic example is the "option base controversy" in the X3J2 committee.*

*Most early versions of BASIC allowed default dimensioning of arrays. That is, in the absence of a DIM statement, the subscripts ofan array (list or table) could range up to a value of 10. But BASICs differed in what they allowed for the lower bound. Some specified the lower bound to be 0, while others specified it to be 1. The argument for 0 is that many elementary applications need the subscript 0, and it should be available for those cases. The argument for 1 is that most arrays naturally begin with 1, and it would be a waste of storage to allow 0 when it isn't needed. The committee argued long and hard over this one. Each time we voted, we tied. The OPTION BASE compromise eventually emerged. The rule is this: if OPTION BASE 0 appears in the program, the lower bound for all subscripts is 0; if OPTION BASE 1 appears, the lower bound is 1; if neither appears, then by default the lower bound is 0.*

*Few members of X3J2 really liked this compromise, but the committee supported it in the interests of getting out the standard for Minimal BASIC. Subsequent efforts to remove this feature failed, for the same reason.*

*In a radical shift, the committee decided, five years later, to allow users to specify lower bounds for individual arrays in dimension statements. Thus, "DIM YEAR(1970:1980)" would create a list named "YEAR" having 11 elements identified with the numbers 1970, 1971. . .1980. Ifno lowerbound is specified, it is assumed to be 1. Both sides now have their wish, but it took more than five years to achieve it.*

## Logistics

*Meetings of standards committees are held near where the members work, and members take tums hosting meetings. Since 1974, X3J2 has met30 times. Ten of these meetings have been in the East, nine in the Midwest and South, and eight in the Far West. Because we are developing the standard jointly with ECMA (European Computer Manufacturers Association) TC21, we hold joint meetings yearly, alternating between the United States and Europe. Three of these meetings have been held in Europe.*

*Rotating meeting sites is required. Often we have to choose between alternate sites based on, for example, availability and cost of accommodations. But one factor we always consider very carefully is food. Whatever site we choose must have good restaurants. Thus we're fortunate that 20 percent of the X3J2 membership works in the San Francisco area. In Europe, the ECMA TC21 members work in ornear London, Paris, and Venice. I don't know what we would have done had computer companies located themselves in remote areas that offered no culinary delights.*

*By and large, the membership of X3J2 has been stable. Representatives from large companies sometimes change, and some members change employers. But, for the most part, the members have known each other and worked together for years. This leads to occasional amusing incidents.*

*In the early days of the committee, before individualized T-shirts became the fad, one memberstood up to speak but instead doffed his shirt to reveal his custom T-shirt that had the words "BASIC Standard" on the front and "Strings Subco" on the back. We now take our special T-shirts for granted, but that one brought down the house.*

*Another member was amused to discover a brand of toilet paper called "Basic"; he presented this as an exhibit to illustrate the then-current status of the standard.*

*More recently, a member of long standing appeared at a meeting carrying a large plastic goose. When the discussion deteriorated (who can be brilliant for six hours a day all week long?), the goose would appear on the table.*