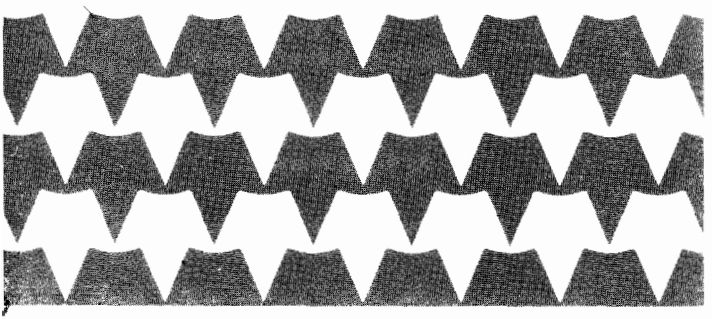
Cat. No. 25-4103

Tandy 3000

BASIC

Quick Reference Guide



TANDY®

Tandy 3000
BASIC Quick Reference Guide

Contents

Syntax

Use the following syntax to load BASIC at the MS-DOS system prompt:

BASIC | **BASICA** [*pathname*] [<*input-file*] [>[>]*output-file*][/F:*max. # of files open at same time*] [/M:*highest memory location for BASIC, max. block size*] [/C:*RS-232 receive buffer size*] [/S:*max. record length for direct access files*][/D][/I]

/D loads the Double Precision Transcendental Math Package.

/I tells BASIC not to dynamically allocate space during file operations. This switch is always invoked if you use BASICA.

Defaults:

/F	=	3 files
/M	=	64K bytes, 4096 blocks of 16 bytes
/C	=	256 bytes
/S	=	128 bytes

Statements and Functions

In the list of statements and functions below, the keyword is bold. Information that:

- Is in upper-case must be typed exactly as is.
- Is in lower-case italics is variable. You supply the value(s).
- Is enclosed in brackets is optional.

ABS(*number*)

Computes the absolute value of *number*.

```
PRINT ABS(-44)        X=ABS(Y)
```

ASC(*string*)

Returns the ASCII code for the first character of *string*.

```
PRINT ASC("A")        N=ASC(B$)
```

ATN(*number*)

Computes the arctangent of *number* in radians.

```
PRINT ATN(7)          X=ATN(Y/3) * 57.29578
```

AUTO [*line*][,*increment*]

Automatically generates a line number each time you press [ENTER], starting with the specified *line* and using the specified *increment*.

AUTO AUTO 100,50

BEEP

Produces a sound at 800 Hz for 1/4 second.

BEEP

BLOAD *pathname* [,*offset*]

Loads a memory image file, specified by *pathname*, into memory. *Offset* is the number of bytes into the current segment at which BASIC loads the image. It must be in the range 0 to 65535. Default = the value set by BSAVE.

BLOAD "prog1.txt"

BLOAD "prog2.txt",0

BSAVE *pathname,offset,length*

Saves the contents of an area of memory as a disk file (*pathname*). *Offset* is the number of bytes into the current segment where BASIC starts saving. It must be in the range 0 to 65535. Default = the value set by BSAVE.

Length is the number of bytes to save. It must be in the range 1 to 65535.

BSAVE "prog1.sav",0,500

CALL *variable* [(*parameter list*)]

Transfers program control to an assembly-language subroutine stored at *variable*. *Parameter list* contains the variables that are passed to the external subroutine.

CALL C CALL C (A$,Z,X)

CALLS *variable* [(*parameter list*)]

Transfers program control to an MS-FORTRAN™ routine stored at *variable*. *Parameter list* is described in CALL.

CALLS X CALLS X (S$)

CDBL(*number*)

Converts *number* to double precision.

PRINT CDBL(465.342) Z=CDBL(A)

CHAIN [MERGE] *pathname* [,[,*line*][,ALL][,DELETE *line-line*]]

Lets the current program load and execute another program (specified by *pathname*). Commas in the syntax line are significant and must be entered even if you omit the option.

Line is the line number at which execution begins in the chained program. Default = the first line of the chained program.

ALL passes every variable in the current program to the chained program. MERGE overlays the lines of the chained program with the current program. DELETE deletes lines in the overlay so that you can merge in a new overlay.

```
CHAIN "prog2"
CHAIN "subprog.bas",,ALL
```

CHDIR *pathname*

Changes the current directory to *pathname*.

```
CHDIR "B:\ACCTS\RECVBLE"
```

CHR$(*code*)

Returns the character corresponding to any ASCII or control *code*.

```
PRINT CHR$(35)        C$=CHR$(32)
```

CINT(*number*)

Converts *number* to integer representation.

```
PRINT CINT(1.6)       Z=CINT(-1.67)
```

CIRCLE [STEP] (*x,y*),*radius*[,*color* [,*start*, [*end* [,*aspect*]]]

Graphics. Draws an ellipse, the center of which is (*x,y*), on the screen. STEP designates (*x,y*) as relative coordinates. *Radius* is the major axis of the ellipse.

Start and *end* are the beginning and ending angles, in radians (in the range −6.283186 to 6.283186). *Aspect* is the ratio of the x-radius to the y-radius in terms of coordinates. See "Text and Graphics Modes" for the range for *color*.

```
CIRCLE (150,100),50
```

CLEAR [,*memory location*] [,*stack space*]

Frees memory for data without erasing the program currently in memory. *Memory location* specifies the highest memory location available for BASIC. *Stack space* specifies the amount of memory to set aside for temporarily storing internal data and addresses.

```
CLEAR        CLEAR ,45000
```

CLOSE [*buffer,...*]

Closes access to a disk file or a communications channel.

```
CLOSE        CLOSE 1,2,8
```

CLS

Clears the screen (or active viewport) and returns the cursor to the home position.

```
CLS
```

COLOR [*background*] [,[*pallette*]]

Graphics. Selects the background color and the palette for Screen Mode 1. *Background* specifies the color of the background. *Palette* specifies the palette to use for the foreground colors. See "Text and Graphics Modes."

```
COLOR 0,7        COLOR 0,1
```

COLOR [*foreground* [,[*background*] [,*border*]]

Text Mode Only. Selects the display colors for the foreground, background, and border for Screen Mode 0. See "Text and Graphics Modes."

```
COLOR 0,7        COLOR 1,0
```

COM(*channel*) *action*

Turns on, turns off, or temporarily halts the trapping of activity on the communications channel. *Channel* can be 1 or 2. *Action* can be ON, OFF, or STOP.

```
COM(1) ON        COM(2) STOP
```

COMMON *variable* [,*variable,...*]

Reserves space for *variables* so they can be passed to a chained program.

```
COMMON A, B$, D()
```

CONT

Resumes execution when a program is stopped by either CTRL BREAK or the execution of a STOP or an END statement.

```
CONT
```

COS(*number*)

Computes the cosine of *number*.

```
PRINT COS(5.8)        Y=COS(X*0.0174533)
```

CSNG(*number*)

Converts *number* to single precision.

```
PRINT CSNG(.145388509)
Z=CSNG(A#)
```

CSRLIN

Returns the current row position of the cursor.

```
PRINT CSRLIN        A=CSRLIN
```

CVD(*8-byte string*)
CVI(*2-byte string*)
CVS(*4-byte string*)

Converts a string to a double precision (CVD), an integer (CVI), or a single precision (CVS) number. Use to restore data to numeric form after it is read from the disk.

```
A# = CVD(GROSSPAY$)
D=CVS(TOTAL$)
```

DATA *constant* [,*constant*,...]

Stores numeric and string constants to be accessed by a READ statement.

```
DATA NEW YORK, CHICAGO, LOS ANGELES
```

DATE$[= *string*]

Sets the date to the specified *string*, or retrieves the current date if you omit *string*.

```
DATE$="04/17/85"        TODAY$=DATE$
```

DEFDBL *letter* [,*letter*,...]
DEFINT *letter* [,*letter*,...]
DEFSNG *letter* [,*letter*,...]
DEFSTR *letter* [,*letter*,...]

Defines any variables beginning with *letter(s)* as double precision (DBL), integer (INT), single precision (SNG), or string (STR).

```
DEFDBL A        DEFINT A-G
```

DEF FN*name* [(*argument list*)] = *expression*

Defines *name* as a function according to the *expression*. *Name* is a valid variable name, and *argument list* is a list of dummy variables used in *expression*. *Expression* defines the operation to be performed.

```
DEF FNR = RND(1)*89+10
```

DEF SEG[= *address*]

Assigns the specified *address* to be the current segment address. *Address* is in the range 0 to 65535. Default = Data Segment.

```
DEF SEG        DEF SEG = &HB800
```

DEF USR[*number*] = *offset*

Defines the user *number* and segment *offset* of a subroutine to be called by the USR function. *Number* is in the range 0 to 9. Default = 0. *Offset* is in the range 0 to 65535.

```
DEF USR = 0        DEF USR3 = &H0020
```

DELETE *line1-line2*

Deletes *line1* through *line2* of the program in memory. *Line1* defaults to the first line of the program, and *line2* defaults to the last line of the program.

```
DELETE 70        DELETE -110
```

DIM *array* (*dimension*)[,*array* (*dimension*),...]

Sets aside storage for *arrays* with the *dimensions* you specify.

```
DIM AR(100)        DIM L1%(8,25)
```

DRAW *string*

Graphics. Draws an image on the screen. *String* specifies 1 or more of the movement commands. Prefix commands can precede the movement commands.

Movement Commands

U[*n*]	Moves up *n* points.
D[*n*]	Moves down *n* points.
L[*n*]	Moves left *n* points.
R[*n*]	Moves right *n* points.
E[*n*]	Moves diagonally up and right *n* points.
F[*n*]	Moves diagonally down and right *n* points.
G[*n*]	Moves diagonally down and left *n* points.
H[*n*]	Moves diagonally up and left *n* points.
M*x,y*	Moves to point *x,y*.

Prefix Commands

B	Plots no points after the move.
N	Returns to the original position when the move is complete.
A*angle*	Sets the angle of the move.
C*color*	Sets the color.
P*color,border*	Paints, using *color* and *border*.
S*factor*	Sets the scale factor.
TA*angle*	Moves at the specified angle.
X*variable*	Executes a substring.

See "Text and Graphics Modes."

```
DRAW "U30;" + "D30;" + "L40;" + "R40;"
```

EDIT *line*

Enter the Edit mode. BASIC displays *line* for editing.

```
EDIT 100
```

END

Ends program execution, and closes all files.

```
END
```

ENVIRON "*parameter id = text*" [;"*parameter id = text*",...]

Advanced Statement. Lets you modify BASIC's Environment String Table.

```
ENVIRON "PATH=A: \"
ENVIRON "SALES=MYSALES"
```

ENVIRON$ [("*parameter id*")][(*number*)]

Advanced Function. Returns the specified environment string from BASIC's Environment String Table. *Number* and *parameter id* are mutually exclusive. Only one can be specified on the command line.

```
PRINT ENVIRON$("PATH")
```

EOF(*buffer*)

Function. Detects the end of a file.

Sequential access files: returns 0 (false) if end-of-file has not been read or -1 (true) if it has.

Direct access files: returns -1 (true) if the last executed GET statement was unable to read an entire record because of an attempt to read beyond the physical end of the file.

```
IF EOF(1) THEN GOTO 1540
```

EOF(*buffer*)

Communications. Detects an empty input queue for a communications file.

ASCII mode: returns -1 (true) if CONTROL-Z is received.
Binary mode: returns -1 (true) when the input queue is empty.

```
IF EOF(1) THEN RETURN
```

ERASE *array* [,*array*,...]

Erases one or more *arrays* from memory.

```
ERASE C        ERASE G,H,I,Z$
```

ERDEV

Advanced Function. Returns the value of a device error within MS-DOS as set by the Interrupt 24 handler.

```
ERDEV
```

ERDEV$

Advanced Function. Returns the name of the device when a device error occurs. A character device error returns an 8-byte device name. A block device error returns a 2-byte device name.

```
ERDEV$
```

ERL

Returns the number of the line in which an error has occurred.

```
PRINT ERL        E = ERL
```

ERR

Returns the error code if an error has occurred.

```
IF ERR = 7 THEN 1000 ELSE 2000
```

ERROR *code*

Simulates a specified error during program execution. *Code* is one of BASIC's error codes.

```
ERROR 1
```

EXP(*number*)

Returns the natural exponent of *number*, that is, *e* (base of natural logarithms) to the power of *number*. *Number* must be less than or equal to 88.02968.

```
PRINT EXP(-2)        A=EXP(-6)
```

FIELD *buffer, length* AS *variable* [,*length* AS *variable*,...]

Divides a direct access buffer into fields so that you can send data from memory to disk and from disk to memory. Each field is identified by a string *variable* and is the *length* you specify. *Length* must be an integer in the range 1 to 255.

```
FIELD 3, 128 as A$, 128 AS B$
```

FILES [*pathname*]

Displays the names of the files and directories on a disk.

```
FILES        FILES "\BOOKS\"
```

FIX(*number*)

Returns the truncated integer of *number*.

```
PRINT FIX(2.6)        Z=FIX(B)
```

FOR *variable = initial value* TO *final value* [STEP *increment*]
NEXT [*variable*]

Establishes a program loop that allows a series of program statements to be executed a specified number of times.

```
FOR I=1 TO 5 : PRINT I : NEXT
```

FRE(*dummy argument*)

Returns the number of bytes in memory not being used by BASIC. Specify a string argument if you want BASIC to compress the data before returning the amount of memory available.

```
PRINT FRE(44)        PRINT FRE("44")
```

GET [#]*buffer* [,*record*]

Reads a record from a direct access disk file, and places it in the specified *buffer*. If you omit *record*, BASIC gets the next sequential record.

```
GET 1        GET 1,25
```

GET [#]*buffer,number*

Communications. Transfers data from the communications line to the communications buffer. *Number* specifies the number of bytes to transfer.

```
GET 1,8
```

GET (*x1,y1*)-(*x2,y2*),*array*

Graphics. Transfers points from an area on the display to an array. (*x1,y1*) and (*x2,y2*) are the coordinates at which the image begins and ends, respectively.

```
GET (0,0) - (100,100),Z
```

GOSUB *line*

Branches to the subroutine, beginning at *line*. Every subroutine must end with a RETURN statement.

```
GOSUB 1000
```

GOTO *line*

Branches to the specified *line*.

```
GOTO 100        IF R = 14 THEN GOTO 80
```

HEX$(*number*)

Calculates the hexadecimal value of *number*.

```
PRINT HEX$(30)        Y$ = HEX$(X/16)
```

IF *expression* **THEN** *statement(s)* [**ELSE** *statement(s)*]

Tests a conditional expression, and makes a decision regarding program flow. If *expression* is true, BASIC executes the THEN *statement*. If *expression* is false, BASIC executes the matching ELSE *statement* or the next program line.

```
IF A < B THEN PRINT "A < B" ELSE PRINT
   "B <= A"
```

INKEY$

Returns a 0-, 1-, or 2-byte string from the keyboard without pressing [ENTER]. INKEY$ does not echo the character to the display.

0-byte string = no key is pressed.

1-byte string = the ASCII value of the key.

2-byte string = 00 in Byte 1, extended ASCII value of the key in Byte 2.

```
10 A$ = INKEY$:IF A$ = "" THEN 10
```

INP(*port*)

Returns the byte read from *port*. *Port* can be any integer from 0 to 65535.

```
100 A = INP(255)
```

INPUT[;] [*"prompt";*]*variable*[,*variable,...*]

Inputs data from the keyboard into one or more variables. BASIC stops execution, displays *prompt* followed by a question mark, and then waits for input. If you do not want BASIC to display the prompt, type a comma, instead of a semicolon, after *prompt*.

If INPUT is immediately followed by a semicolon, BASIC does not echo the [ENTER] key when you press it as part of a response.

```
INPUT Y%
INPUT "ENTER YOUR NAME AND AGE";
   N$, A
```

INPUT# *buffer, variable*[,*variable...*]

Inputs data from a sequential device or file, and stores it in a program *variable*.

```
INPUT#1, A,B        INPUT#4, A$, B$, C$
```

INPUT$(*number* [,[#]*buffer*])

Inputs a string of characters from either the keyboard or a sequential access file. *Number* specifies the number of characters to be input. It can be in the range 1 to 255.

If you include *buffer*, BASIC inputs the string from a sequential access file. If you omit *buffer*, BASIC inputs the string from the keyboard.

```
A$ = INPUT$(5)        A$ = INPUT$(11,3)
```

INSTR([*number*,]*string1*,*string2*)

Searches for the first occurrence of *string2* in *string1*, and returns the position at which the match is found. *Number* specifies the position in *string1* at which to begin searching. Default = the first character in *string1*.

```
INSTR (3, "1232123", "12")
A$ = "LINCOLN":INSTR(A$,"INC")
```

INT(*number*)

Converts *number* to the largest integer that is less than or equal to *number*. *Number* is not limited to the integer range.

```
PRINT INT(79.89)        PRINT INT (-12.11)
```

IOCTL [#]*buffer*,*string*

Advanced Statement. Sends a control data string to a device driver. *String* is a string expression containing a series of commands. The commands are separated by semicolons. *String* can be a maximum of 255 bytes.

```
IOCTL #1,"PL56"
```

IOCTL$([#]*buffer*)

Advanced Function. Returns the control data string from a device driver that you have opened previously.

```
IF IOCTL$(1) = "NR" THEN PRINT "PRINTER
   NOT READY"
```

KEY *number*,*string*

Assigns or displays function key values. *Number* indicates the function key (1-10) or the user key (15-20) being defined. See KEY (*number*) *action*.

String is the string expression assigned to the key. It can contain a maximum of 15 characters.

KEY ON

Displays the function key assignment values on Line 25 of the screen.

KEY OFF

Erases the soft key assignments from Line 25.

KEY LIST

Displays all 15 characters of all 10 soft key assignments on the screen.

KEY(*number*) *action*

Turns on, turns off, or temporarily halts key trapping for a specified key. *Action* can be ON, OFF, or STOP.

Number can be a number in the range 1 to 20, indicating the number of the key to trap. Function keys use their corresponding function key number (1-10). The cursor direction keys are numbered 11-14. User-defined keys are numbered 15-20. Use the following syntax to define your own user keys:

KEY *number*, CHR$(*key*) + CHR$(*scan*)

KILL *pathname*

Kills (deletes) *pathname* from disk.

KILL "file.bas" KILL "A:\REPORT\data"

LCOPY

Copies all text data on the screen to the printer.

LCOPY

LEFT$(*string,number*)

Returns the specified number of characters from the left portion of *string*. *Number* must be an integer in the range 1 to 255.

PRINT LEFT$("BATTLESHIPS", 6)

LEN(*string*)

Returns the number of characters in *string*. Blanks are counted.

X = LEN(SENTENCE$)

PRINT LEN("DOG") + LEN("TERRIER")

[**LET**] *variable* = *expression*

Assigns the value of *expression* to *variable*.

LET A$ = "A ROSE IS A ROSE"

B1 = 1.23

LINE [(*x1,y1*)]-(*x2,y2*),[*color*][,B[F]][,*style*]

Graphics. Draws a line or a box on the video display.

(*x1,y1*) is the point at which the line begins. Default = last point referenced on the screen.

(*x2,y2*) is the point at which the line ends.

Color indicates the color of the line. See "Text and Graphics Modes."

With the **B** option, BASIC draws a box. The points that you specify are opposite corners. If you specify both the **B** and **F** options, BASIC draws a box and fills the box in with *color*.

Style is a 16-bit integer that lets you select the line-style used when drawing normal lines and unfilled boxes. If bit = 1, the point is drawn. If bit = 0, the point is not drawn.

```
LINE (0,0)-(319,199)        LINE -(319,199)
```

LINE INPUT[;]["*prompt*";] *string variable*

Inputs an entire line (a maximum of 254 characters) from the keyboard including delimiters (commas, quotations marks, etc.). The only way to terminate the string input is to press [ENTER]. However, if LINE INPUT is immediately followed by a semicolon, pressing [ENTER] does not echo a carriage return to the display.

```
LINE INPUT A$

LINE INPUT "LAST NAME, FIRST NAME? ";
  N$
```

LINE INPUT# *buffer, variable*

Inputs an entire line of data from a sequential file including delimiters (commas, quotation marks, etc).

```
LINE INPUT#1, A$
```

LIST [*startline*][-[*endline*]][,*device*]

Lists a program in memory to the display or specified *device*.

```
LIST        LIST 50-100,"LPT1:"
```

LLIST [*startline*][-[*endline*]]

Lists program lines in memory to the printer. LLIST assumes an 80-character-wide printer. (See WIDTH.)

```
LLIST       LLIST 68-90
```

LOAD *pathname* [,R]

Loads a BASIC program from disk into memory. The R option tells BASIC to run the program.

```
LOAD "A:prog1.bas"
LOAD "prog1.bas",R
```

LOC(*buffer*)

Returns the current record position within a file.

Direct access file: returns the record number accessed by the last GET or PUT statement.

Sequential access file: returns the number of 128-byte records that have been read or written.

```
IF LOC(1)>55 THEN END
```

LOC(*buffer*)

Communications. Returns the number of characters in the input queue.

If more than 255 characters are in the input queue, LOC returns 255. If fewer are there, LOC returns the actual number of characters waiting to be read.

```
IF LOC(X)>0 THEN 1000
```

LOCATE [*row*][,[*column*][,[*cursor*][,[*start*][,*stop*]]]]

Positions the cursor on the screen at the position indicated by *row* and *column*.

Cursor indicates whether the cursor is visible or invisible. 1 = visible cursor, and 0 = invisible cursor.

Start is a numeric expression in the range 0 to 31 that specifies the first scan line of the cursor. *Stop* specifies the last scan line of the cursor, and it can be in the same range as *start*.

```
LOCATE 10,20,1,4        LOCATE 25,1,1,7
```

LOCK [#]*buffer* [,*record*]
UNLOCK [#]*buffer* [,*record*]

Controls access by other processes to all or part of an opened file, specified by *buffer*. LOCK and UNLOCK are used only by the compiler. *Record* is the record or the range of records to lock or unlock.

```
LOCK 1, 1 TO 4        UNLOCK 1, 1 TO 4
```

LOF(*buffer*)

Returns the length of the file in bytes.

```
Y = LOF(5)
```

LOF(*buffer*)

Communications. Returns the amount of free space in the input queue.

```
IF LOF(X) < 20 GOTO 1000
```

LOG(*number*)

Computes the natural logarithm of *number*. *Number* must be greater than zero.

```
PRINT LOG(3.14159)
Z = 10 * LOG(P5/P1)
```

LPOS(*number*)

Returns the logical position of the print head within the printer's buffer. *Number* can be 0 or 1 to indicate LPT1:.

```
100 IF LPOS(X)>60 THEN LPRINT
```

LPRINT [USING *format*;] *data*[,*data*,...]

Prints *data* on the printer. LPRINT assumes a print width of 80 characters. (See WIDTH.)

See PRINT and PRINT USING for more information on formatting the output.

```
LPRINT (A * 2)/3
LPRINT USING "#####.#"; 2.17
```

LSET *field name* = *data*

Moves *data* to the direct access buffer, and places it in *field name*, in preparation for a PUT statement. *Field name* is a string variable defined in a FIELD statement. LSET left-justifies the data.

```
LSET AD$ = "2000 EAST PECAN ST."
```

MERGE *pathname*

Loads a BASIC program and merges it with the program currently in memory. The file must be in ASCII format. (It must have been saved with the A option).

```
MERGE "prog2.txt"
```

MID$(*oldstring*,*start* [,*length*]) = *newstring*

Replaces a portion of *oldstring* with *newstring*. *Start* specifies the position of the first character you want to change. *Length* specifies the number of characters you want to replace.

```
MID$ (A$,3,4) = "12345": PRINT A$
```

MID$(*string, start* [,*length*])

Returns a substring of *string*. *Start* specifies the position in the string from which to get the substring. *Length* is the number of characters in the substring. It must be in the range 1 to 255.

```
A$ = "WEATHERFORD":PRINT MID$(A$, 3,
   2)
```

MKDIR *pathname*

Creates the directory specified by *pathname*.

```
MKDIR "A:\ACCTS\PAYABLE"
MKDIR "\ADDRESS"
```

MKD$(*double precision expression*)
MKI$(*integer expression*)
MKS$(*single precision expression*)

Converts numeric values to string values. MKD$ returns an 8-byte string, MKI$ a 2-byte string, and MKS$ a 4-byte string. These are the inverse functions of CVD, CVI, and CVS.

```
LSET YTD$ = MKS$(564.33)
LSET TOT$ = MKS$(TOT)
```

NAME *old filename* AS *new filename*

Renames *old filename* as *new filename*.

```
NAME "file.bas" AS "file.old"
```

NEW

Deletes the program currently in memory, and clears all variables.

```
NEW
```

OCT$(*number*)

Returns a string that represents the octal value of a decimal *number*.

```
PRINT OCT$(30)        S$=OCT$(90)
```

ON COM(*channel*) **GOSUB** *line*

Transfers program control to a subroutine beginning at *line* when activity occurs on the specified communications *channel*.

```
ON COM(1) GOSUB 1000
```

ON ERROR GOTO *line*

Transfers control to *line* if an error occurs. You must execute an ON ERROR GOTO before the error occurs.

```
10 ON ERROR GOTO 1500
```

ON *n* **GOSUB** *line*[,*line*,...]

Looks at *n* and branches to the subroutine at the *n*th line listed in the statement.

For example, if *n* equals 1, BASIC branches to the first line listed. If *n* equals 2, BASIC branches to the second line listed. *N* must be in the range 0 to 255.

```
ON Y GOSUB 1000, 2000, 3000
```

ON *n* **GOTO** *line*[,*line*,...]

Goes to the *line* specified by the value of *n*.

For example, if *n* equals 1, BASIC branches to the first line listed. If *n* equals 2, BASIC branches to the second line listed. *N* must be in the range 0 to 255.

```
ON MI GOTO 150, 160, 170, 150, 180
```

ON KEY(*number*) **GOSUB** *line*

Transfers program control to a subroutine beginning at *line* when you press the specified key.

Number indicates the number of the key to trap. Function keys are numbered 1-10. The cursor direction keys are numbered 11-14. User keys are numbered 15-20. User keys are defined with the KEY statement.

```
ON KEY(13) GOSUB 500
```

ON PEN GOSUB *line*

Transfers program control to the subroutine at *line* when you activate the light pen.

```
ON PEN GOSUB 1000
```

ON PLAY(*number*) **GOSUB** *line*

Transfers program control to the subroutine beginning at *line* when the number of notes in the background music buffer goes from *number* to *number* minus 1. *Number* must be in the range 1 to 32.

```
ON PLAY(30) GOSUB 200
```

ON STRIG(*number*) **GOSUB** *line*

Transfers program control to the subroutine at *line* when you press one of the joystick's buttons.

Integer specifies the button pressed and is one of the following:

0	left joystick, button 1
2	right joystick, button 1
4	left joystick, button 2
6	right joystick, button 2

```
10 ON STRIG(0) GOSUB 1000
```

ON TIMER(*number*) **GOSUB** *line*

Transfers program control to the subroutine at *line* when the specified period of time has elapsed. *Number* indicates the number of seconds. It can be a value in the range 1 to 86400 (86400 seconds = 24 hours).

```
ON TIMER(3600) GOSUB 500
```

OPEN *mode,buffer,*[*pathname*][*dev:*][,*record length*]

OPEN [*pathname*][*dev:*] [FOR mode] [*access*] AS *buffer* [LEN = *record length*]

Establishes an input/output path for a file or device.

Buffer specifies the I/O buffer in memory to use when accessing the file. It can be in the range 1 to 255.

If you do not specify *pathname*, you must specify *dev:*.

Record length sets the record length for direct access files. It can be in the range 2 to 32768. Default = 128 bytes.

Mode specifies any of the following:

O	or	OPEN	sequential output mode
I	or	INPUT	sequential input mode
A	or	APPEND	sequential extension of an existing file
R	or	RANDOM	direct input/output mode

In the first form of the syntax, you must use the abbreviated form of *mode*, and must enclose it in quotation marks.

In the second form of the syntax, you must specify the complete word for *mode*. You cannot specify RANDOM. If you want to use direct access in the second form of the syntax, omit *mode*.

Access controls the processes that can access the file and the degree to which they do so. *Access* can be SHARED, LOCK READ, LOCK WRITE, or LOCK READ WRITE.

```
OPEN "R",2,"test.dat"
OPEN "LPT1:" FOR OUTPUT AS 2
```

OPEN "COM *channel*: [*speed*] [,*parity*] [,*data*][,*stop*][,RS] [,CS[*seconds*]][,DS[*seconds*]] [,CD[*seconds*]][,*mode*][,PE][,LF]" [FOR *mode*] AS [#][*buffer*][LEN = *number*]

Communications. Opens a file and allocates a buffer for RS-232C (Asynchronous Communications Adapter) communication.

Channel can be 1 or 2 to select the communications channel to be opened.

Speed specifies the baud rate. It can be 75, 110, 150, 300, 600, 1200, 2400, 4800, or 9600. Default = 300.

Parity can be E for EVEN, O for ODD, M for MARK, S for SPACE or N for NO. Default = E.

Data specifies the number of bits. It can be 5, 6, 7, or 8. Default = 7.

Stop can be either 1 or 2 to indicate the number of stop bits. Default = 2 for baud rates of 75 and 100, and 1 for all other baud rates.

Mode is either OUTPUT or INPUT for sequential access. Default = random input/output.

Buffer indicates the buffer that accesses the file. It can be in the range 1 to 15.

Number specifies the maximum number of bytes that can be accessed in the communications buffer by GET and PUT statements. Default = 128 bytes.

```
OPEN "COM1:" AS 1
OPEN "COM1:9600,N,8,1,BIN" AS 2
```

OPTION BASE *value*

Sets *value* as the minimum value for an array subscript. This statement must precede the DIM statement. *Value* can be 1 or 0. Default = 0.

```
OPTION BASE 1
```

OUT *port, data byte*

Sends a *data byte* to a machine output *port*. *Port* is an integer in the range 0 to 65535, and *data byte* is an integer in the range 0 to 255.

```
OUT 32,100
```

PAINT (*x,y*) [*color*[,*border*][,*background*]]

Graphics. Fills an area on the display with a selected color or pattern.

(*x,y*) are the coordinates at which painting begins.

Color can be either a number or a string expression. If color is a number, it specifies a color number available in the current screen mode. (See "Text and Graphics Modes.") If color is a string expression, it specifies the mask to be used for tiling in the form:

CHR$(&H*nn*) + CHR$(&H*nn*) + CHR$(&H*nn*)...

Border specifies the color of the border of the object. Default = *color*.

Background is the color to skip when checking for borders while paint tiling.

PEEK(*memory location*)

Returns a byte from *memory location* . *Memory location* must be in the range -32768 to 65535.

```
A = PEEK (&H5AØØ)
```

PEN(*number*)

Returns the light pen's coordinates.

Number is a number in the range 1 to 9 that tells BASIC what to return.

Ø	Returns -1 if the pen button has been pressed since the last poll. Returns Ø if not.
1	Returns the x-coordinate (horizontal) at which the pen was last activated.
2	Returns the y-coordinate (vertical) at which the pen was last activated.
3	Returns -1 if the pen button is pressed. Returns Ø if it is up.
4	Returns the last known valid x-coordinate (horizontal).
5	Returns the last known valid y-coordinate (vertical).
6	Returns the character row position at which the pen was last activated.
7	Returns the character column position at which the pen was last activated.
8	Returns the last known character row position.
9	Returns the last known character column position.

```
A = PEN(1)
```

PEN *action*

Turns on, turns off, or temporarily halts light pen event trapping.

Action can be ON, OFF, or STOP.

```
PEN ON        PEN OFF        PEN STOP
```

PLAY *string*

Plays musical notes specified by *string.*

String is a string expression consisting of 1 or more single-character music commands.

Single-Character Music Commands

A-G plays notes A through G of one musical scale. Include a number sign (#) or plus sign (+) to indicate a sharp note. Include a minus sign (-) to indicate a flat note.

L*n* sets the duration of the notes that follow. *N* can be a value in the range 1 to 64:

1 indicates a whole note.
2 indicates a half note.
4 indicates a quarter note.
8 indicates an eighth note.
16 indicates a sixteenth note.

O*n* sets the current octave. There are 7 octaves, Ø-6. Octave 3 starts with middle C. Default = Octave 4.

N*n* plays a note. *N* can be in the range Ø to 84.

P*n* rests. *N* can be in the range 1 to 64.

T*n* sets the number of quarter notes in 1 minute. *N* can be in the range of 32 to 255. Default = 12Ø quarter notes in 1 minute.

. plays as a dotted note. BASIC plays the note one half its length longer.

MF plays the music in the foreground. Default = MB.

MB plays the music in the background. A maximum of 32 notes and/or rests can play in background at a time. Default = MB.

MN sets "music normal." Each note plays 7/8 of the duration as set by the L option. Default = MN.

ML sets "music legato." Each note plays the full duration as set by the L option. Default = MN.

MS sets "music staccato." Each note plays 3/4 of the duration as set by the L option. Default = MN.

X *variable*; executes a substring. You can have one string execute another, which executes a third, and so on.

```
1Ø PLAY "C4F.C8F8.C16F8.G16A2F2"
```

PLAY(*number*)

Returns the number of notes currently in the background music queue. *Number* is a dummy argument.

```
X=PLAY(Ø)        X=PLAY(2)
```

PLAY *action*

Turns on, turns off, or temporarily halts background music event trapping.

Action can be ON, OFF, or STOP.

```
PLAY ON        PLAY OFF        PLAY STOP
```

PMAP(*coordinate,action*)

Returns the physical or world coordinate for the specified coordinate.

Coordinate is any x- or y-coordinate.

Action is one of the following:

0	returns the physical x-coordinate for the specified world coordinate.
1	returns the physical y-coordinate for the specified world coordinate.
2	returns the world x-coordinate for the specified physical coordinate.
3	returns the world y-coordinate for the specified physical coordinate.

```
X = PMAP(200,0)      Z = PMAP(50,0)
```

POINT (*x,y*)
POINT (*action*)

Returns the color number of a point on the screen, or returns the current physical or world coordinates. (*x,y*) are the coordinates of the point.

Action is one of the following:

0	returns the current physical x-coordinate (horizontal).
1	returns the current physical y-coordinate (vertical).
2	returns the world x-coordinate if WINDOW is active. Otherwise, returns the physical x-coordinate.
3	returns the world y-coordinate if WINDOW is active. Otherwise, returns the physical y-coordinate.

```
IF POINT(1,1) <> 0 THEN PRESET(1,1)
   ELSE PSET(1,1)
```

POKE *memory location, data byte*

Writes *data byte* into *memory location*. Both *memory location* and *data byte* must be integers. *Memory location* must be in the range −32768 to 65535.

```
10 POKE &H5A00, &HFF
```

POS(*number*)

Returns the current column position of the cursor.

Number is a dummy argument.

```
IF POS(X) > 70 THEN IF A$ = CHR$(32)
   THEN A$ = CHR$(13)
```

PRINT *data*[,*data*, . . .]

Prints numeric or string *data* on the display. If you use commas, the cursor automatically advances to the next tab position before printing the next item. If you use semicolons or spaces to separate the data items, PRINT prints the items without any spaces between them.

```
PRINT "DO"; "NOT"; "LEAVE"; "SPACES";
    "BETWEEN"; "THESE"; "WORDS"
PRINT "THE TOTAL IS",TTL
```

PRINT USING *format; data*[,*data*, . . .]

Prints data using a format you specify.

Format consists of 1 or more field specifier(s), or any alphanumeric character.

Specifiers for String Fields:

!	prints the first character in the string only.
\spaces\	prints 2 + *n* characters from the string. (*N* is the number of spaces between the slashes.)
&	prints the string without modifications.

Specifiers for Numeric Fields:

#	prints the same number of digit positions as number signs (#).
+	prints the sign of the number.
–	prints a negative sign *after* negative numbers (and a space after positive numbers).
**	fills leading spaces with asterisks.
$$	prints a dollar sign immediately before the number.
**$	fills leading spaces with asterisks, and prints a dollar sign immediately before the number.
,	prints a comma before every third digit to the left of the decimal point.
^^^^	prints in exponential format.
_	prints the next character as a literal character.

```
PRINT USING ".####^^^^"; 888888
PRINT USING "**$###,.##"; 1234.5
```

PRINT# *buffer*,[USING *format*] *data*[,*data* . . .]

Writes data items to a sequential access disk file. See PRINT USING.

```
PRINT# 1,A        PRINT# 1, B$,T$
```

PSET (*x,y*)[,*color*]
PRESET (*x,y*)[,*color*]

Graphics. Draws a point on the display. (*x,y*) are the coordinates of the point. *Color* specifies the color of the point.

If you use PSET, *color* defaults to the foreground color. If you use PRESET, *color* defaults to the background color.

See "Text and Graphics Modes."

```
PSET (1,1)        PRESET (1,1),0
```

PUT [#]*buffer*[,*record*]

Puts a *record* in a direct access disk file. *Record* is the record number to be written to the file and must be in the range 1 to 16,777,215. Default = the current record number.

```
PUT 4         PUT 1,25
```

PUT [#]*buffer,number*

Communications. Transfers data from the communications buffer to the communications line. *Number* is the number of bytes to transfer.

```
PUT 2,80
```

PUT (*x,y*),*array*[,*action*]

Graphics. Transfers an image stored in an array onto the screen.

(*x,y*) are the coordinates at which the image begins (the upper left corner of the image). Default = the last point referenced.

Array is the array variable name that holds the image.

Action sets the type of interaction between the transferred image and the image already on the screen. *Action* can be PSET, PRESET, AND, OR, or XOR. Default = PRESET.

```
PUT (200,100),A
```

RANDOMIZE [number]

Reseeds the random number generator. *Number* can be an integer or a single- or double-precision number.

```
RANDOMIZE      RANDOMIZE 300
RANDOMIZE TIMER
```

READ *variable*[,*variable*, . . .]

Reads values from a DATA statement, and assigns them to *variables*.

```
READ T        READ N$
```

REM

Inserts a remark line in a program. You can use an apostrophe (') as an abbreviation for REM.

```
REM AVERAGE VELOCITY        'TOTALS
```

RENUM [*new line*][,[*line*][,*increment*]]

Renumbers the program currently in memory including line numbers appearing after GOTO, GOSUB, THEN, ON/GOTO, ON/GOSUB, ON ERROR GOTO, RESUME, and ERL.

Line is the line in the program at which BASIC starts renumbering. Default = the first line.

New line is the new line number assigned to *line*. Default = 10.

Increment tells BASIC how to number the successive lines. Default = 10.

```
RENUM        RENUM 600, 5000, 100
```

RESET

Closes all open files on all drives.

```
RESET
```

RESTORE [*line*]

Restores a program's access to previously read DATA statements. *Line* specifies the DATA statement to be accessed at the next READ statement. Default = the first DATA statement.

```
RESTORE
```

RESUME [*line*]
RESUME NEXT

Resumes program execution after an error-handling routine.

RESUME *line* branches to the specified line number. Default = the statement in which the error occurred. RESUME NEXT branches to the statement following the point at which the error occurred.

```
RESUME        RESUME 10
RESUME NEXT
```

RETURN [*line*]

Returns control from a subroutine executed by a GOSUB to the specified *line*. Default = the line immediately following the GOSUB.

```
RETURN        RETURN 40
```

RIGHT$(*string, number*)

Returns the specified number of characters from the far right portion of *string*. *Number* must be an integer in the range 1 to 255.

```
PRINT RIGHT$("WATERMELON",5)
PRINT RIGHT$("PUPPY",25)
```

RMDIR *pathname*

Removes (deletes) the directory specified by *pathname*. The directory being deleted must be empty except for the "." and ".." symbols. See the MS-DOS COPY and KILL commands.

```
RMDIR "NAMES"
RMDIR "A:\ACCTS\PAYABLE"
```

RND(*number*)

Returns a random number in the range Ø to 1.

Number is an integer in the range −32767 to 32768. If *number* is negative, RND starts the sequence of random numbers at the beginning. If *number* is Ø, RND repeats the last number generated. If *number* is Ø or a positive number, RND returns the next number in the sequence.

```
PRINT RND(1)        A = RND(2)
```

RSET *field name = data*

Sets *data* in a direct access buffer *field name*, in preparation for a PUT statement, and right-justifies it. See LSET.

RUN [*line*]
RUN *pathname*[,R]

Executes a program. *Line* is the program line at which BASIC begins execution. Default = the first line.

If you specify the R option, BASIC does not close the open files before loading the new program into memory.

```
RUN          RUN 1ØØ          RUN "program.a"
```

SAVE *pathname* [,A]
SAVE *pathname* [,P]

Saves a program on disk with the specified name. The A option saves the program in ASCII format. Default = the compressed format.

```
SAVE "A:file1.bas"
SAVE "\EDUC\mathpak.txt", A
```

SCREEN (*row, column,*[*number*])

Returns the ASCII code for the character at the specified row and column. If *number* is specified and is non-zero, BASIC returns the color attribute.

```
A = SCREEN(20,20)
PRINT SCREEN(10,10,1)
```

SCREEN [*mode*][,[*burst*][,[*active page*][,*display page*]]]

Sets the screen attributes to be used by all other graphics statements.

Mode is an integer in the range 0 to 2 that specifies the screen mode.

Burst activates or de-activates color in Screen Mode 0 or 1.
In Mode 0: Use 1 to activate, and 0 to de-activate.
In Mode 1: Use 0 to activate, and 1 to de-activate.

Active page selects the video page to which BASIC is to write. Default = the current active page.

Display page selects the video page BASIC is to display. Default = the current active page.

```
10 SCREEN 0,0        60 SCREEN 2
```

SGN(*number*)

Determines *number*'s sign. If *number* is negative, SGN returns -1. If *number* is positive, SGN returns 1. If *number* is 0, SGN returns 0.

```
Y = SGN(A * B)
```

SHELL [*command*]

Advanced Statement. Loads and executes another program (.EXE or .COM) or an internal command as a child process to the original program.

Command is a string expression containing the name of the program you want to run.

```
SHELL
```

SIN(*number*)

Returns the sine of *number*. *Number* must be in radians.

```
PRINT SIN(7.96)
```

SOUND *frequency,duration*

Generates a sound with the *frequency* and *duration* specified.

Frequency is an integer in the range 37 to 32767, indicating the frequency in Hertz.

Duration is a numeric expression in the range 0.027 to 65535, specifying the duration in clock ticks.

```
SOUND 37,2
```

SPACE$(*number*)

Returns a string of *number* spaces.

```
PRINT "COST" SPACE$(4) "QUANTITY"
   SPACE$(9) "TOTAL"
```

SPC(*number*)

Skips *number* spaces in a PRINT statement.

```
PRINT "HELLO" SPC(15) "THERE"
```

SQR(*number*)

Returns the square root of *number*. *Number* must be greater than zero.

```
PRINT SQR(155.7)
```

STICK (*action*)

Returns the coordinates of the joysticks.

Action can be 0 or 1 for the horizontal and vertical coordinates of the left joystick, respectively. It can be 2 or 3 for the horizontal and vertical coordinates of the right joystick, respectively. You must read 0 before you can read 1, 2, and 3.

```
A=STICK(0)
```

STOP

Stops program execution.

```
STOP
```

STR$(*number*)

Converts *number* to a string.

```
S$ = STR$(X)        PRINT STR$(-234)
```

STRIG ON
STRIG OFF

Enables the STRIG function. STRIG ON lets you execute STRIG function statements to return the status of the joystick buttons. If you execute a STRIG OFF statement, you cannot execute a STRIG function statement.

```
STRIG ON        STRIG OFF
```

STRIG(*number*)

Function. Returns the status of joystick buttons. You must execute STRIG ON before using this function.

Number is a number in the range 0 to 7 to test the status of the joystick buttons. Even numbers test to see if the joystick button has been pressed and released. Odd numbers test to see if the button is currently being pressed.

0,1 Left joystick, Trigger 1.

2,3 Right joystick, Trigger 1.

4,5 Left joystick, Trigger 2.

6,7 Right joystick, Trigger 2.

```
IF STRIG(0) THEN BEEP
```

STRIG(*number*) *action*

Turns on, turns off, or temporarily halts joystick trapping. *Action* is ON, OFF, or STOP.

Number specifies the joystick and trigger. It can be:

0 Left joystick, Trigger 1.

2 Right joystick, Trigger 1.

4 Left joystick, Trigger 2.

6 Right joystick, Trigger 2.

```
STRIG(0) ON
```

STRING$(*number,character*)

Returns a string containing the specified number of *character.*

Number must be in the range 0 to 255.

Character is a string or an ASCII code.

```
B$ = STRING$(25, "X")
PRINT STRING$(50, 10)
```

SWAP *variable1,variable2*

Exchanges the values of two variables of the same type.

```
SWAP F1#, F2#
```

SYSTEM

Returns you to the MS-DOS command level.

```
SYSTEM
```

TAB(*number*)

Spaces to position *number* on the display. *Number* must be in the range 1 to 255.

```
PRINT "NAME" TAB(25) "AMOUNT":PRINT
```

TAN(*number*)

Returns the tangent of *number*. *Number* must be in radians.

```
PRINT TAN(7.96)        S = TAN(X)
```

TIME$[= *string*]

Sets the time to the specified *string* or retrieves the current time if you omit *string*. BASIC uses a 24-hour clock. *String* is a literal, enclosed in quotation marks.

```
TIME$ = "14:15"        PRINT TIME$
```

TIMER

Returns the number of seconds since midnight or since the last system reset.

```
A = TIMER        PRINT TIMER
```

TIMER *action*

Turns on, turns off, or temporarily halts timer event trapping. *Action* can be ON, OFF, or STOP.

```
TIMER ON        TIMER OFF
TIMER STOP
```

TROFF
TRON

Turns on (TRON) or turns off (TROFF) the program flow tracer.

```
TRON        TROFF
```

USR[*number*](*argument*)

Calls a user's assembly-language subroutine identified with *number*, and passes *argument* to that subroutine.

The *number* you specify must be the same as the corresponding DEF USR statement for that routine. Default = 0.

VAL(*string*)

Calculates the numerical value of *string*.

```
PRINT VAL("100")        PRINT VAL("1234E5")
```

VARPTR (*variable*)
VARPTR ([#]*buffer*)

Returns the offset into BASIC's data segment of a variable of the file control block. When used with *variable*, VARPTR returns the address of the first byte of data identified with *variable*. When used with *buffer*, it returns the address of the file's control block.

```
A = VARPTR(A$)        PRINT VARPTR(3)
```

VARPTR$(*variable*)

Returns a 3-byte string representing a memory address of a variable. Byte 0 is the *type*, Byte 1 is the *low byte of address*, and Byte 2 is the *high byte of address*.

```
PLAY "X" + VARPTR$(A$)
```

VIEW [SCREEN] [(*x1,y1*)-(*x2,y2*)[,[*color*][,[*border*]]]]

Graphics. Creates a viewport that redefines the screen parameters.

(*x1,y1*) specifies the upper-left coordinates for the rectangular viewport.

(*x2,y2*) specifies the lower-right coordinates for the rectangular viewport.

SCREEN specifies that all coordinates used in drawing are absolute to point 0,0 on the screen. If you omit SCREEN, all coordinates specified are relative to the viewport coordinates.

```
VIEW (10,10)-(100,100)
VIEW SCREEN (20,25)-(100,150)
```

VIEW PRINT *top line* TO *bottom line*

Creates a text viewport that redefines the text screen parameters.

Top line specifies the first line of the text viewport. It can be in the range 1 to 24, but must be less than *bottom line*. Default = Line 1.

Bottom line specifies the last line of the text viewport. It can be in the range 1 to 24, but must be greater than *top line*. Default = Line 24.

```
VIEW 1 TO 15
```

WAIT *port, number1* [,*number2*]

Suspends program execution until a machine input *port* develops a specified bit pattern. *Number1* and *number2* are integers in the range 0 to 255.

```
100 WAIT 32,2
```

WHILE *expression*
WEND

Executes a series of statements in a loop as long as a given condition is true.

If *expression* is true, BASIC executes the statements after the WHILE statement until it encounters a WEND statement. If *expression* is still true, BASIC repeats the process. If it is not true, execution resumes with the statement following the WEND statement.

WIDTH [LPRINT] *size*
WIDTH *buffer, size*
WIDTH *device, size*

Sets the line width in number of characters for the display, line printer, or communications channel.

Size can be an integer in the range 0 to 255 that specifies the number of characters in a line. For the screen, *size* can be only 40 or 80. Default = 255 for the communications channel.

```
WIDTH 40        WIDTH LPRINT 100
WIDTH "SCRN:", 40
```

WINDOW [SCREEN] [(*x1,y1*)-(*x2,y2*)]

Lets you change the physical coordinates of the screen (or current viewport) by defining "world" coordinates.

(*x1,y1*) are the world coordinates for the upper-left corner of the screen.

(*x2,y2*) are the world coordinates for the lower-left corner of the screen.

The SCREEN option tells BASIC to set the coordinates like the screen display. If you omit screen, BASIC inverts the y-coordinates to show a true Cartesian coordinate system.

WINDOW lets you plot points outside the normal screen coordinate limits by setting new world coordinates to the screen.

```
WINDOW (1984,100000)-(1987,300000)
```

WRITE *data*[,*data*, . . .]

Outputs data to the screen.

WRITE# *buffer, data*[,*data*, . . .]

Writes *data* to a sequential access disk file.

```
WRITE#1, A$,B$
```

Function Key Settings

F1	LIST"	F6	,"LPT1:" [ENTER]
F2	RUN [ENTER]	F7	TRON [ENTER]
F3	LOAD"	F8	TROFF [ENTER]
F4	SAVE"	F9	KEY
F5	CONT [ENTER]	F10	SCREEN 0,0,0 [ENTER]

Typing Keywords Using The [ALT] Key

A	AUTO	N	NEXT
B	BSAVE	O	OPEN
C	COLOR	P	PRINT
D	DELETE	Q	(none)
E	ELSE	R	RUN
F	FOR	S	SCREEN
G	GOTO	T	THEN
H	HEX$	U	USING
I	INPUT	V	VAL
J	(none)	W	WIDTH
K	KEY	X	XOR
L	LOCATE	Y	(none)
M	MOTOR *	Z	(none)

* MOTOR is a reserved word, but not recognized in this implementation of BASIC.

Exponential Notation and Numeric Precision Characters

D	Used in double precision exponential notation.
E	Used in single precision exponential notation.
%	Makes the variable preceding it integer precision.
!	Makes the variable preceding it single precision.
#	Makes the variable preceding it double precision.
$	Makes the variable preceding it a string.

Operator Precedence

Each operator or group of operators takes precedence over the group below it.

( )	Parentheses
^	Exponentiation
+ -	Unary positive, negative
* /	Multiplication, division
\	Integer division
MOD	Modulus arithmetic
< > <=	Relational tests
>= <>	
NOT	
AND	
OR	
XOR	
EQV	
IMP	

Text and Graphics Modes

Screen Mode 0 = 16-Color Text Mode (40 or 80 columns).

Foreground can be any of the following 16 colors, in either non-blinking mode (0-15) or blinking mode (16-31). For blinking color, add 16 to the number given below.

Background can any of Colors 0 - 7. *Border* can be any of Colors 0 - 15.

No.	Color	No.	Color
0	black	8	gray
1	blue	9	light blue
2	green	10	light green
3	cyan	11	light cyan
4	red	12	light red
5	magenta	13	light magenta
6	brown	14	yellow
7	white	15	high-intensity white

Screen Mode 1 = 4-Color Graphics Mode (320 x 200).

Foreground can be any of the colors listed below (in the current palette). *Background* can be any of the 16 colors listed for Screen Mode 0.

Palette 0		Palette 1	
No.	Color	No.	Color
0	*C. B. G.*	0	*C. B. G.*
1	green	1	cyan
2	red	2	magenta
3	brown	3	high-intensity white

C.B.G. = Current Background Color

Screen Mode 2 = Black-and-White Graphics Mode (640 x 200).

Foreground is white and *background* is black.

No.	Color
0	black
1	white

Error Codes and Messages

Number	Message
1	NEXT without FOR
2	Syntax error
3	RETURN without GOSUB
4	Out of DATA
5	Illegal function call
6	Overflow
7	Out of memory
8	Undefined line number
9	Subscript out of range
10	Redimensioned Array/Duplicate Definition
11	Division by zero
12	Illegal direct
13	Type mismatch
14	Out of string space
15	String too long
16	String formula too complex
17	Can't continue
18	Undefined user function
19	No RESUME
20	RESUME without error
21	Unprintable error
22	Missing operand
23	Line buffer overflow
24	Device Timeout
25	Device Fault
26	FOR without NEXT
27	Out of Paper
29	WHILE without WEND
30	WEND without WHILE
50	FIELD overflow
51	Internal error
52	Bad file number
53	File not found
54	Bad file mode
55	File already open
57	Device I/O error
58	File already exists
61	Disk full
62	Input past end
63	Bad record number
64	Bad file name
66	Direct statement in file
67	Too many files
68	Device Unavailable
69	Communication buffer overflow

Number	Message
70	Disk write protect
71	Disk not Ready
72	Disk media error
73	Advanced Feature
74	Rename across disks
75	Path/File Access Error
76	Path not found
77	Deadlock
78	Unprintable Error

RADIO SHACK, A Division of Tandy Corporation

U.S.A.: FORT WORTH, TEXAS 76102
CANADA: BARRIE, ONTARIO L4M 4W5

AUSTRALIA	BELGIUM	FRANCE	U. K.
91 Kurrajong Avenue Mount Druitt, N.S.W. 2770	Rue des Pieds d'Alouette, 39 5140 Naninne (Namur)	BP 147-95022 Cergy Pontoise Cedex	Bilston Road Wednesbury West Midlands WS10 7JN

07/86-TP 875-9931 Printed in U.S.A.