TANDY®

GW-BASIC

Quick Reference

GW-BASIC
Quick Reference

10 9 8 7 6 5 4 3 2 1

Contents

Introduction 1

Loading GW-BASIC 2

GW-BASIC Commands and Statements . . 3

GW-BASIC Function Key Settings 34

Typing Keywords Using the ALT Key . . . 34

Exponential Notation and Numeric Precision Characters 35

Operator Precedence 35

Text and Graphics Modes 36

4-Color Set, 1 Palette 38

4-Color Set, 2 Palettes 39

16-Color Set 39

Enhanced Graphics Color Selection 39

Error Codes and Messages 40

Introduction

This quick reference presents only a brief description of each GW-BASIC function. The *BASIC Reference Manual* for your computer provides more detailed information.

Notations:
The following typeface conventions and notations are used in this quick reference.

BOLD UPPERCASE indicates a command or statement. Type commands and statements exactly as they appear.

lowercase italics represent variable names, letters, characters, or values. You supply the names, letters, characters, or values.

[ ] (brackets) indicate optional parameters. Include these optional parameters if you want the functions they provide.

... (ellipsis) indicates that you can repeat a parameter.

(Graphics) indicates that you must be in a graphics screen mode to use a particular function. The abbreviation EGA indicates that a function is available only if your system has an enhanced graphics adapter.

| between two items indicates that either of the items, but not both, can be included in a command.

SMALL CAPS BOLD indicates a key, such as **ENTER**, that you press.

Loading GW-BASIC

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

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

Pathname Loads and executes the specified GW-BASIC program file.

<input file Inputs data from a file instead of the keyboard.

output file Outputs data to a file instead of the display. Use > to overwrite the existing output file or > > to append to it.

/F:*files* The maximum number of files (0-15) that can be open at the same time during the current GW-BASIC program. If /F is used, /I must also be used.

/M:*workspace, blocksize* GW-BASIC reserves *(workspace* x 16) bytes, *blocksize* of which are reserved for the program. Any excess bytes are reserved for programs or machine language routines.

/C:*RS-232 receive buffer size* Defaults to 256 bytes. The RS232 transmit buffer is always 128 bytes.

/S: *max.record length for direct access files* Defaults to 128 bytes. Maximum is 32767 bytes.

/D Loads a double-precision math package.

/I Prevents dynamic allocation of file space. Automatically invoked with BASICA.

Defaults:

/F:3	/M:64000, 4096
/C:256	/S:128

GW-BASIC
Commands and Statements

ABS(*number*)

Computes the absolute value of *number*.

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

ASC(*string*)

Returns ASCII code for first character of *string*.

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

ATN(*number*)

Computes the arctangent of *number* radians.

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

AUTO [*starting line number*] [,*increment*]

Generates program line numbers. To turn off AUTO, press **BREAK**.

```
AUTO          AUTO 100,50
```

BEEP

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

```
IF X > 20 THEN BEEP
```

BLOAD *pathname* [,*offset*]

Loads memory image *pathname* into memory, *offset* (0-65535) bytes from top of current segment.

```
BLOAD "prog1.txt"     BLOAD "prog2.txt",0
```

BSAVE *pathname*,*offset*,*length*

Saves *length* bytes (1-65535), beginning *offset* bytes (0-65535) from top of current segment, as *pathname*.

```
BSAVE "prog1.sav",0,500
```

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

Executes external subroutine stored at *variable* and passes *parameter list* to the subroutine.

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

CDBL(*number*)

Converts *number* to double precision.

```
PRINT CDBL(465.342)          Z=CDBL(A)
```

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

Chains *pathname* to the current program, beginning execution of *pathname* at *startline*. 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. ALL permits access to current variables and arrays.

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

CHDIR *pathname*

Sets *pathname* as the current directory.

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

CHR$(*code*)

Returns the character equivalent of an ASCII or control *code*.

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

CINT(*number*)

Rounds *number* to the nearest integer.

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

CIRCLE [STEP] (*xcenter*, *ycenter*), *radius*[,*color*[,*start*,[*end*[,*aspect*]]]]

(Graphics) Draws an ellipse. *Start* and *end* are in radians (-6.283186 to 6.283186). *Aspect* is the ratio of the x-radius to the y-radius in terms of coordinates. STEP tells BASIC that the (*xcenter*, *ycenter*) coordinates are relative to the last point referenced.

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

CLEAR [*,memsize*] [*,stacksize*]
[*,video memory*]

Frees memory and initializes variables without erasing the program in memory. *Video memory* (Tandy 1000 family only) specifies the amount of memory to set aside.

```
CLEAR   CLEAR ,45000   CLEAR ,6100,300
```

CLOSE [*buffer,...*]

Closes access to *buffer*. Default = All.

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

CLS

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

```
CLS
```

COLOR [*background*] [*,foreground palette*]

(Graphics) Sets background color and foreground palette for Screen Mode 1. See "Text and Graphics Modes."

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

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

(Text Mode) Sets the display colors for Screen Mode 0. See "Text and Graphics Modes."

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

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

(Graphics) Sets foreground and background colors for Screen Modes 3-6 (Tandy 1000 family only). See "Text and Graphics Modes."

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

(EGA) Sets foreground color (identified by palette position) and background color (0-15) for Screen Modes 7-10. See "Text and Graphics Modes."

```
COLOR 7,0
```

COM(*channel*) *action*

Controls event trapping on *channel*. *Channel* can be 1 or 2. *Action* is ON, OFF, or STOP.

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

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

Allows access to variables by chained programs. Use COMMON in each program.

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

CONT

Resumes execution after **CTRL-BREAK**, STOP, or END.

```
CONT
```

COS(*number*)

Computes the cosine of *number* radians.

```
PRINT COS(5.8)        Y=COS(X*.00174533)
```

CSNG(*number*)

Converts *number* to single precision and rounds to 7 significant figures.

```
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 *string* to double-precision (CVD), integer (CVI), or single-precision (CVS) form.

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

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

Stores *constants* for access by a READ statement.

```
DATA 2,4,6,8
DATA "NEW JERSEY","DALLAS,TEXAS"
```

DATE$[=*string*]

Sets or retrieves the current date.

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

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

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

```
DEFDBL A          DEFINT A,C,H-M
```

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

Defines function *name*, which uses *argument list* (a list of dummy variables) to perform *expression*.

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

DEF SEG[=*address*]

Assigns *address* (0-65535) to be the current segment address. Default = Data Segment address.

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

DEF USR[*number*] =*offset*

Defines the user *number* (0-9) and segment *offset* (0-65535) of an assembly language subroutine to be called by the USR function.

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

DELETE [*line1*] [-[*line2*]]

Deletes *line1-line2* of the current program. Deletes all lines if you omit both parameters.

```
DELETE 70         DELETE -110
```

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

Allocates storage for *array(s)*.

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

DRAW *string*

(Graphics) Draws the image defined by the movement/prefix commands in *string*. Movement is relative to current graphics position. (Default = center of screen.) Movement can be U (up), D (down), L (left), R (right), E (up and right), F (down and right), G (down and left), or H (up and left), followed by the number of points to move. See also "Text and Graphics Modes."

```
DRAW "U30 D30 L40 R40"
```

EDIT *line*

Enters Edit mode and displays *line* for editing.

```
EDIT 100      EDIT .
```

END

Ends program execution and closes all files.

```
END
```

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

Sets the Environment String Table value of *parameter id* to *text*. *Parameter id* must be typed in all uppercase.

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

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

Returns the string named *parameter id* or the string in position *number* from GW-BASIC's Environment String Table.

```
PRINT ENVIRON$("PATH")
PRINT ENVIRON$(3)
```

EOF(*buffer***)**

(Function) Detects the end of a file. True is indicated by a value of -1, false by a value of 0.

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

EOF(*buffer*)

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

ASCII mode: true (–1) if CONTROL-Z is received.

Binary mode: true (–1) if input queue is empty.

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

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

Erases array(s) from memory. Lets you redimension arrays or use their previously allocated space for other purposes.

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

ERDEV

Returns an MS-DOS device error as set by the Interrupt 24 handler. Lower 8 bits of ERDEV contain the Interrupt 24 error code.

```
ERDEV
```

ERDEV$

Returns a device name, as set by the Interrupt 24 handler, when a device error occurs. Character device names contain 8 bytes. Block device names contain 2 bytes.

```
ERDEV$
```

ERL

Returns the number of the line in which an error occurred, or 0 if no error occurred.

```
PRINT ERL              E=ERL
```

ERR

Returns the error code if an error occurred.

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

ERROR *code*

Simulates a specified GW-BASIC error *code*.

```
ERROR 1
```

EXP(*number*)

Calculates 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)
```

EXTERR (*number*)

Returns extended information about an error code, class, recovery or location, when *number* is 0, 1, 2 or 3, respectively.

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

Divides a direct access *buffer* into fields. Each field is assigned a size in bytes (*length*) and a name (*variable*). *Length* is an integer, 1-255.

```
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*)

Truncates *number* to an integer.

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

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

Initializes *variable* to *initial value* and executes *statements* until *variable* reaches *final value*. *Increment* can be a positive or negative integer; the default value is 1. *Variable* must be either integer or single precision.

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

FRE(*dummy argument*)

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

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

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

Reads *record* from a direct access disk file and places it in *buffer*. *Record* is an integer in the range 0-16,777,215.

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

GET[#]*buffer*,*number*

Transfers *number* bytes from communications line to communications buffer.

```
GET 1,8
```

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

(Graphics) Transfers points from (*x1*,*y1*) to (*x2*,*y2*) into a numeric *array*.

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

GOSUB *line*

Branches to the subroutine beginning at *line*. Subroutines must end with RETURN.

```
GOSUB 1000
```

GOTO *line*

Branches to *line*.

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

HEX$(*number*)

Calculates hexadecimal value of *number*. *Number* is a decimal, -32768 to 65535.

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

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

Executes THEN *statement(s)* if *expression* is true. Executes ELSE *statement(s)* or next program line if *expression* is false.

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

INKEY$

Returns the first character in the keyboard buffer, or returns the null string if the buffer is empty. The character is not displayed.

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

INP(*port***)**

Reads a byte from *port*. *Port* is an integer, 0-65535.

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

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

Allows for data entry from the keyboard during program execution. GW-BASIC stops execution and displays *prompt* and a question mark. Use a comma instead of a semicolon after *prompt* if you do not want the question mark displayed. The semicolon after INPUT does not echo **ENTER** when you press it as part of the response.

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

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

Inputs data from *buffer* (a sequential device or file) and stores it in *variable*.

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

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

Inputs *number* characters (1-255) from the keyboard or a sequential access *buffer*.

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

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

Searches for *string2* in *string1*, beginning at position *number* or the first character in *string1*. Returns the position at which the first match is found.

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

INT(*number*)

Truncates *number* to integer form.

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

IOCTL [#]*buffer*,*string*

Sends control data *string* to a device driver. Use semicolons to separate the commands in *string*. *String* can be a maximum of 255 bytes.

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

IOCTL$([#]*buffer*)

Returns the control data string from *buffer*, a previously opened device driver.

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

KEY *number*,*string*

Assigns function *string* (up to 15 characters) to function key *number*. Function keys are numbered 1-10; user keys are numbered 15-20.

KEY ON

Displays soft key assignments on screen line 25.

KEY OFF

Erases the soft key assignments from screen 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. *Number* can be 1-10 for function keys, 11-14 for cursor direction keys, or 15-20 for user-defined keys. *Action* can be ON, OFF, or STOP.

Use the following syntax to define user keys:

```
KEY number,CHR$(key hexcode) +
    CHR$(scan code)
```

KILL *"pathname"*

Kills (deletes) *pathname* from disk.

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

LEFT$(*string,number*)

Returns *number* (1-255) characters from the left portion of *string.*

```
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 *expression* to *variable.*

```
LET A$ = "A ROSE IS A ROSE"
B1 = 1.23
```

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

(Graphics) Draws a *color* line (or box if the B option is included) from (*x1, y1*) to (*x2, y2*) on the display. The F option fills the box. See "Text and Graphics Modes." *Style* is a 16-bit integer mask.

```
LINE (0,0)-(319,199)
LINE (0,0)-(319,199),,,&HF0F0
```

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

Inputs up to 255 characters from the keyboard. **ENTER** designates the end of the line. A semicolon following LINE INPUT prevents **ENTER** from echoing a carriage return to the display.

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

LINE INPUT# *buffer, variable*

Inputs a line of data from sequential file *buffer* into *variable*.

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

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

Lists *startline* to *endline* of the current program to *device* (SCRN: or LPT1:).

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

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

Lists *startline* to *endline* to the printer. Assumes a 132-character-wide printer. See the WIDTH statement to change this setting.

```
LLIST         LLIST 68-90
```

LOAD "*pathname*" [,R]

Loads *pathname*. R option runs the program.

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

LOC(*buffer*)

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 read or written.

```
A=LOC(2)    IF LOC(1) > 55 THEN END
```

LOC(*buffer*)

(Communications) Returns the number of characters in input queue *buffer*. Returns 255 if *buffer* contains more than 255 characters.

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

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

Positions cursor on the screen at *row*, *column*. Cursor is visible if *cursor* = 1, invisible if *cursor* = 0. *Start scan line* and *stop scan line* (both 0-31) control cursor size.

```
LOCATE 10,20,1,4
LOCATE 25,1,1,3
```

LOCK [#]*buffer*[,*record1* [**TO** *record2*]]
UNLOCK [#]*buffer*[,*record 1*[**TO** *record2*]]

Controls access by other processes to all or part of open *buffer*. Use with the MS-DOS **share** command.

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

LOF(*buffer*)

Returns the length of *buffer* in bytes.

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

LOF(*buffer*)

(Communications) Returns the amount of free space in the input queue *buffer*.

```
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*...]

Sends *data* to the printer. Assumes 80-character print width. See also WIDTH, PRINT, and PRINT USING.

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

LSET *field name = data*

Moves *data,* left justified, to *field name* , in the direct access buffer, in preparation for a PUT statement. Use FIELD before LSET to define fields. Any numeric value placed in a direct access file buffer with LSET must be converted to a string. See MKS$, MKD$, and MKI$.

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

MERGE *pathname*

Loads *pathname* (an ASCII-format program file) and merges it with the program in memory.

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

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

Replaces *length* characters of *oldstring* with *newstring,* beginning at position *start.*

```
MID$ (A$,3,4) = "1234":PRINT A$
```

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

Returns a substring of *length* characters from *string,* beginning at position *start. Length* can be 1-255.

```
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 strings. MKD$ returns an 8-byte string, MKI$ a 2-byte string, and MKS$ a 4-byte string. 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 current program; clears all variables.

```
NEW
```

OCT$(*number***)**

Returns octal equivalent of decimal *number*.

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

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

Transfers to subroutine at *line* when activity occurs on *channel* (1 or 2). *Line* = 0 turns off trapping.

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

ON ERROR GOTO *line*

Transfers to *line* if an error occurs. You must execute an ON ERROR GOTO before the error occurs. *Line* = 0 turns off error trapping.

```
10 ON ERROR GOTO 1500
```

ON *number* **GOSUB** *list*

Transfers to the subroutine beginning at the line specified as item *number* in *list*. *Number* can be 0-255. *List* is a series of line numbers. If *number* = 1, the program branches to the first entry in *list*. If *number* = 2, the program branches to the second entry in *list*.

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

ON *number* **GOTO** *list*

Transfers to the line specified as item *number* in *list*. *Number* can be 0-255. *List* is a series of line numbers. If *number* = 1, the program branches to the first entry in *list*. If *number* = 2, the program branches to the second entry in *list*.

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

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

Transfers to a subroutine beginning at *line* if KEY *number* is pressed. Function keys are numbered 1-10; cursor direction keys, 11-14; user keys 15-20. Use KEY to define user keys.

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

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

Transfers to the subroutine at *line* when the number of notes in the background music buffer goes from *number* to *number* minus 1. *Number* can be 1-32.

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

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

Transfers to the subroutine at *line* when you press joystick button *number*. *Number* = 0 (left joystick, button 1), 2 (right joystick, button 1), 4 (left joystick, button 2), or 6 (right joystick, button 2). *Line* = 0 turns off joystick trapping.

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

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

Transfers to the subroutine at *line* when *number* seconds have elapsed. *Number* can be 1-86400 (24 hours).

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

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

Creates an input/output path for a sequential file or device. *Buffer* specifies the I/O buffer (1-the maximum number of files allowed) in memory to use when accessing the file. Use a single letter in quotation marks for *mode*: O (Output), I (Input), A (Append), or R (Random Input/Output). R is the default. *Record length* cannot exceed the value specified with the GW-BASIC /s: switch. Default = 128 bytes.

```
OPEN "R",2,"TEST.DAT"
```

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

Creates an input/output path for a random-access file or device. Use OPEN, INPUT, or APPEND for *mode*. RANDOM is not valid. Do not enclose *mode* in quotation marks, and do not use a single letter only.To use direct access in this syntax, omit *mode*

Access can be SHARED, LOCK READ, LOCK WRITE, or LOCK READ WRITE.

```
OPEN "LPT1:"FOR OUTPUT AS 2.
```

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

(Communications) Opens *channel* (1 or 2) and allocates *buffer* for RS-232C (asynchronous) communication. *Speed* is 75, 110, 150, 300, 600, 1200, 2400, 4800, or 9600 (bps). Default = 300. *Parity* is E, O, M, S, or N (for even, odd, mark, space, or no parity checking). Default = E. Data is 5, 6, 7 or 8 (transmit and receive bits). Default = 7. *Stop* is 1 or 2 (stop bits). *Data type* is BIN (binary) or ASC (ASCII). Default = BIN. *Number* is the maximum number of bytes that can be accessed in the communications buffer by GET and PUT statements. Default = 128.

```
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. *Value* can be 1 or 0; default = 0. OPTION BASE must precede the DIM statement.

```
OPTION BASE 1
```

OUT *port, data byte*

Sends *data byte* to *port*. *Port* is an integer, 0-65535. *Data byte* is an integer, 0-255.

```
OUT 32,100
```

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

(Graphics) Fills a screen area from (*x*,*y*) to the nearest borders on the display with *color* or a pattern.

Color is a currently valid color number, or a masking string for titling, in the form:

```
CHR$(&Hnn)+CHR$(&Hnn)=CHR$(&Hnn)...
```

(See "Text and Graphics Modes.")
Border is the border color. Default = *color*.
Background allows re-painting without erasing previous layers.

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

(Graphics, EGA) In an EGA environment, GW-BASIC stores the string as a stack of 8-bit units, called bit planes.

PALETTE [*color number*,*display color*]

(Graphics; any Tandy business computer with EGA, any Tandy 1000 family computer.) Associates *display color* with *color number* in the current palette.

```
PALETTE 3,7
```

PALETTE [*palette position*] [,*color number*]

(Graphics, EGA) Change one or more of the colors in the color palette.

```
PALETTE 3,7
```

PALETTE USING *array*(*subscript*)

(Graphics; any Tandy business computer with EGA, any Tandy 1000 family computer.) Assigns new colors to all 16 slots of the current palette. New values are taken from *array*, an integer array, beginning with position *subscript*. *Array* must include enough entries beyond *subscript* to fill the palette.

```
PALETTE USING A(0)
PALETTE USING A(2)
```

PCOPY *source page*, *destination page*

Copies video *source page* to *destination page*.

```
PCOPY 3,5          PCOPY 6,4
```

PEEK(*memory location*)

Returns a byte from *memory location*. *Memory location* can be 0-65535. The value returned is an integer in the range 0-255.

```
A = PEEK(&H5A00)
```

PLAY *string*

Plays musical notes specified by *string*. *String* is a series of single-character music commands.

Commands are:

A-G	Notes. Optional # or + indicates sharp note. Optional - indicates flat note.
L*n*	Sets duration of notes. *N* = 1 (whole), 2 (half), 4 (quarter), etc.
O*n*	Sets octave *n* (0-6).
>(<)	Goes up (down) an octave.
N*n*	Plays note (0-84).
P*n*	Rests (1-64). See L for values.
T*n*	Sets number of quarter notes per minute (32-255).
.	Plays a dotted note.
MF (MB)	Plays music in foreground (background).
MN (ML, MS)	Sets music normal (legato, staccato).

X*variable* Executes a substring.
V*n* Sets volume (0-15).

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

PLAY(*number*)

Returns the number of notes in the background music queue. The maximum returned value is 32 (the maximum buffer size). *Number* is the voice channel (0, 1, or 2) on Tandy 1000 family computers and a dummy argument on Tandy business computers.

```
X=PLAY(0)  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 a physical or world x- or y-coordinate for a mapped x- or y-*coordinate*. *Action* is 0, 1, 2, or 3, indicating that *coordinate* is a physical x, physical y, world x, or world y, respectively.

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

POINT(*x, y*)

(Graphics) Returns color number of (*x,y*)

```
IF POINT(1,1)=0 THEN PRESET
```

POINT(*action*)

Returns the current physical or world coordinates of the cursor. *Action* is 0, 1, 2, or 3, indicating that *coordinate* is a physical x, physical y, world x, or world y, respectively.

```
X=POINT(0)
```

POKE *memory location, data byte*

Writes *data byte* to *memory location*. Use hexadecimal form for *data byte* and *memory location*. *Memory location* must be in the range 0-65535.

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

POS(*number*)

Returns current column position of the cursor. *Number* is a dummy argument.

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

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

(Graphics) Draws a *color* point at (*x, y*) on the display. STEP indicates that (*x,y*) are relative coordinates. *Color* defaults to the foreground color in PSET and to the background color in PRESET. See "Text and Graphics Modes."

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

PRINT *data*[[,|;]*data*...]

Prints numeric or string *data* on the display. Use commas to tab between items. Use semicolons or spaces to print without spaces.

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

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

Prints *data* using the specified *format*. *Format* consists of one or more field specifier(s) or alphanumeric character(s) in quotation marks. *Data* can be a string, a numeric value, or both.

Field specifiers are:

!	Prints first character only
\spaces\	Prints 2 + *n* characters, where *n* is number of spaces typed
&	Prints string as is
#	Prints same number of digits as number signs
+	Prints number's sign
-	Prints a negative sign or space, as appropriate, after number
**	Fills leading spaces with asterisks
$$	Prints $ before number
**$	Fills leading spaces with asterisks and prints $
,	Prints commas in numbers > 999

^ ^ ^ ^ Prints in exponential format

_ Prints next character as literal character (instead of a specifier)

```
PRINT USING ".####^^^^"; 888888
PRINT USING "###.##"; 876.567
```

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

Writes *data* to sequential access *buffer*. Does not compress data before writing to disk. Produces ASCII-coded image of data. See PRINT USING for *format*.

```
PRINT#1,A              PRINT#1, B$, T$
PRINT#1,USING "###.##";A(T)
```

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

See PRESET.

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

Puts *record* (1-16,777,215) in direct access *buffer*. Default = current record number.

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

PUT [#]*buffer, number*

Transfers *number* bytes from the communications buffer to the communications line.

```
PUT 2,80
```

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

(Graphics) Transfers a graphics image from *array* to the screen. (*x, y*) is the upper left corner of the image. Default = the last point referenced.

Action can be PSET, PRESET, AND, OR or XOR. Default = PSET.

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

RANDOMIZE *number*
RANDOMIZE TIMER

Reseeds the random number generator. *Number* can be an integer or a single- or double-precision number. RANDOMIZE TIMER seeds the random number generator without prompting.

```
RANDOMIZE TIMER
RANDOMIZE 300
```

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

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

```
READ T        READ N$, D$, T
```

REM

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

```
REM AVERAGE VELOCITY
'TOTALS
```

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

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

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

RESET

Closes all open files on all drives.

```
RESET
```

RESTORE [*line*]

Restores access to *line*, a previously read DATA statement. Default = the first DATA statement.

```
RESTORE                RESTORE 600
```

RESUME [*line*]
RESUME NEXT

Resumes program execution after an error-handling routine. Execution resumes at *line* or the NEXT statement after 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 rightmost *number* characters of *string*. *Number* is an integer, 1-255.

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

RMDIR *directory*

Removes (deletes) *directory*. This directory must be empty except for the "." and ".."symbols. See the KILL command.

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

RND[(*number*)]

Returns a random number in the range 0-1. *Number* is an integer in the range -32767 to +32768. RND starts the sequence of random numbers again (if *number* is negative), repeats the last number generated (if *number* is 0), or returns the next number (if number is positive).

```
PRINT RND(1)          A = RND(0)
```

RSET *field name = data*

Right-justifies *data* in a direct access buffer *field name*, in preparation for a PUT.

```
RSET A$ = CVI(Z)
```

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

Executes the program currently in memory. Execution begins at *line*. Default = first line. R option leaves current data files open.

```
RUN          RUN 100
```

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

Executes the program identified by *pathname*. R option leaves current data files open for access by new program.

```
RUN "program.a"
```

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

Saves current program as *pathname*. The A option saves in ASCII format; P saves in protected format. Default (no options) = compressed format.

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

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

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

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

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

Sets the screen mode and attributes for all other graphics statements. (CIRCLE, LINE, DRAW, and so on). *Mode* is 0-10. See "Text and Graphics Modes." *Burst* (0 or 1) enables/disables color in Mode 0, 1, or 4.

```
SCREEN 0,0          SCREEN 2
SCREEN 8,1
```

SGN(*number*)

Determines *number*'s sign. Returns -1 if *number* is negative, 1 if *number* is positive, and 0 if *number* is 0.

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

SHELL [*command*]

Loads and executes another program (with an .exe or a .com extension), an internal command, or a batch file as a child process to the original program. *Command* is a string containing the name of the program to run.

```
SHELL "FORMAT B:"
```

SIN(*number*)

Returns the sine of *number* radians.

```
PRINT SIN(7.96)        D=SIN(T)
```

SOUND *frequency,duration*

Generates *frequency* Hertz sound for *duration* clock ticks. *Frequency* is an integer, 1-32767 (Tandy 1000 family computers) or 37-32767 (Tandy business computers). *Duration* is a numeric expression, 1-65535, specifying the number of clock ticks.

```
SOUND 37,2
```

SPACE$(*number*)

Returns a string of *number* spaces.

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

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* = 0 or 1 for horizontal or vertical coordinate of the left joystick. *Action* = 2 or 3 for horizontal or vertical coordinate of the right joystick. Read 0 before reading 1, 2, or 3.

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

STOP

Stops program execution.

```
STOP
```

STR$(*number*)

Converts *number* to a string.

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

STRIG (*action*)

Enables and disables STRIG (*number*) functions. *Action* can be ON, OFF or STOP. STRIG ON must be executed before any other STRIG functions can be executed.

```
STRIG ON
```

STRIG(*number*)

Returns the status of the joystick buttons. Values 0, 2, 4, and 6 see whether button has been pressed since last STRIG function. Values 1, 3, 5, and 7 see whether button is being pressed. STRIG returns -1 for yes, 0 for no. See ON STRIG for button numbers. Execute STRIG ON before using this function.

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

STRIG(*number*) *action*

Turns on, turns off, or temporarily halts joystick trapping. *Action* is ON or OFF. *Number* specifies the joystick and button. See ON STRIG.

```
STRIG(0) ON STRIG(6) OFF
```

STRING$(*number,character*)

Creates a string in which *character* is repeated *number* times. *Number* is 0-255. *Character* is a string or an ASCII code.

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

SWAP *variable1,variable2*

Exchanges values of two variables of same type.

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

SYSTEM

Returns control to MS-DOS command level.

```
SYSTEM
```

TAB(*number*)

Spaces to position *number* on the display. *Number* can be 1-255.

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

TAN(*number*)

Returns the tangent of *number* radians.

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

TIME$[=*string*]

Sets *string* to current time (24-hour clock), or gets the current time if you omit *string*.

```
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* is 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 the user-defined subroutine identified by *number*. Passes *argument* to the subroutine. *Number* (0-9) must be the same as the corresponding DEF USR statement for that routine. Default = 0. *Argument* is a numeric or string expression passed to the subroutine.

VAL(*string*)

Calculates the numerical value of *string*.

```
PRINT VAL("100")      PRINT VAL(NUM$)
```

VARPTR(*variable*)

Returns address of first byte of *variable*. The address is the offset from the data segment.

VARPTR([#]*buffer*)

Returns address of *buffer*'s control block. The address is the offset from the data segment.

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

VARPTR$(*variable*)

Returns a 3-byte string representing the memory address of *variable*. Byte 0 is the variable type (2 for integer, 3 for string, 4 for single-precision, 8 for double-precision), Byte 1 is the low byte of address, and Byte 2 is the high byte of address.

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

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

(Graphics) Creates a rectangular viewport that redefines the screen parameters. Measures coordinates relative to (0, 0) if SCREEN is specified, or relative to viewport if SCREEN is not specified. Optionally fills the viewport with *color*. (*x1*, *y1*) is the upper-left coordinate of the viewport. (*x2*, *y2*) is the lower-right coordinate of the viewport.

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

VIEW PRINT *top line* **TO** *bottom line*

Creates a text viewport. *Top line* (1-24) is the first line of the text viewport. It must be less than *bottom line* (1-24); Default = 1. *Bottom line* is the last line of the text viewport. Default = 24.

```
VIEW PRINT 1 to 15
```

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

Suspends program execution until *port* develops a specified bit pattern. *Number1* and *number2* are integers, 0-255. BASIC XORs data with *number2* or 0, then ANDs result with *number1*. If result = 0, BASIC reads data again. If not, it continues with next statement.

```
WAIT 32,2
```

WHILE *expression*
statement(s)
WEND

Executes *statement(s)* as long as logical *expression* is true. If *expression* is not true, execution resumes with the statement following the WEND statement.

```
WHILE A
PRINT "Calculating..."
WEND
```

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

Sets the line width for the display, printer, or communications channel, to *size* characters. For the screen, *size* can be 40 or 80. Default = 255 (communications channel). *Device* can be SCRN:, LPT2:, COM1:, or COM2:.

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

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

Changes physical coordinates of the screen or current viewport. Plots points outside normal screen coordinate limits by setting new world coordinates to the screen.

(*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.

SCREEN sets coordinates like the screen display. If you omit SCREEN, GW-BASIC inverts the y-coordinates to show a true Cartesian coordinate system.

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

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

Outputs *data* to the screen.

```
10 A=80:B=90:C$='That'sAll'
20 WRITE A,B,C$
```

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

Writes *data* to a sequential access disk file.

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

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

†MOTOR is a reserved word, but it is not recognized in this implementation of GW-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
+, -	Addition, subtraction
<, >, < =, > =, < >, =	Relational tests
NOT	
AND	
OR	
XOR	
EQV	
IMP	

Text and Graphics Modes

Screen mode availability depends on the computer type and the presence of an EGA adapter:

Tandy 1000 Family Computers	
With EGA	Screen Modes 0-2, 7-10
Without EGA	Screen Modes 0-6
Tandy Business Computers	
With EGA	Screen Modes 0-2, 7-10
Without EGA	Screen Modes 0-2

Use the SCREEN statement to select the screen mode.

Screen Mode 0:
16-color text
40 or 80 columns

Use COLOR [*foreground*][,*background*] [,*border*] to select the foreground color (0-31), the background color (0-7), and the border color (0-15). See "16-Color Set."

Screen Mode 1:
4-color graphics
320 x 200

One of two 4-color palettes is available at a time. Use COLOR [*background*][,*foreground palette*] to select the foreground palette (0 or 1) and the background color (0-15). See "4-Color Set, 2 Palettes." Use PALETTE [*palette position*][,*display color*] to change the color associated with *palette position*. (In Screen Mode 1, PALETTE can be used only with Tandy 1000 family computers or Tandy business computers with EGA.)

Screen Mode 2:
Black & white graphics
640 x 200

Foreground is white; background is black.

Screen Mode 3:
16-color graphics
160 x 200

Use COLOR [*foreground*][,*background*] to select the foreground color (0-15) and the background and border color (0-15). See "16-Color Set."

Screen Mode 4:
4-color graphics
320 x 200

Use COLOR [*foreground*][,*background*] to select the foreground color (0-3) and the background and border color (0-15). See "4-Color Set, 1 Palette" for valid *foreground* values and "16-Color Set" for valid *background* values.

Screen Mode 5:
16-color graphics
320 x 200

See Screen Mode 3 description.

Screen Mode 6:
4-color graphics
640 x 200

See Screen Mode 4 description.

Screen Mode 7:
16-color graphics
320 x 200

Sixteen of 64 colors are available at a time. Use COLOR [*foreground palette position*] [,*background*] to select the foreground palette position (0-15) and the background and border color (0-15) to display. See "Enhanced Graphics Color Selection" for valid *foreground palette position* values and "16-Color Set" for valid *background* values. Use PALETTE [*palette position*][,*color number*] to associate a *color* (0-63) with a *palette position* (0-15).

Screen Mode 8:
16-color graphics
640 x 200

See Screen Mode 7 description.

Screen Mode 9:
16-color graphics
640 x 350

Use COLOR [*foreground palette position*] [,*background*] to select the foreground palette position (0-15) and the background border color (0-63) to display. See "Enhanced Color Graphics Selection."

Screen Mode 10:
Black & white graphics
640 x 350

Monochrome monitor required. Use COLOR [*foreground*][,*background*] to select *foreground* (1-3, where 1 = normal intensity, 2 = blinking, 3 = high-intensity) and *background* (0-8). *Background* values are:

0 = off
1 = blinking off to on
2 = blinking off to high-intensity
3 = blinking on to off
4 = on
5 = blinking on to high-intensity
6 = blinking high-intensity to off
7 = blinking high-intensity to on
8 = high-intensity

4-Color Set, 1 Palette

0 = *background*	2 = magenta
1 = cyan	3 = high-intensity white

Background is the current background color.

4-Color Set, 2 Palettes

Palette 0	Palette 1
0 = *background*	0 = *background*
1 = green	1 = cyan
2 = red	2 = magenta
3 = brown	3 = high-intensity white

16-Color Set

Colors 0-15 are non-blnking. For blinking color, add 16 to a number below.

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

Enhanced Graphics Color Selection

The following 64 colors (eight shades of eight colors) are available with an EGA/EGM combination in enhanced mode:

Color	Shades							
Black	0	8	16	24	32	40	48	56
Blue	1	9	17	25	33	41	49	57
Green	2	10	18	26	34	42	50	58
Cyan	3	11	19	27	35	43	51	59
Red	4	12	20	28	36	44	52	60
Magenta	5	13	21	29	37	45	53	61
Yellow	6	14	22	30	38	46	54	62
White	7	15	23	31	39	47	55	63

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

Number	Message
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
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

RADIO SHACK
A Division of Tandy Corporation
Fort Worth, Texas 76102

6/89- TP 875-8330 Printed in U.S.A.